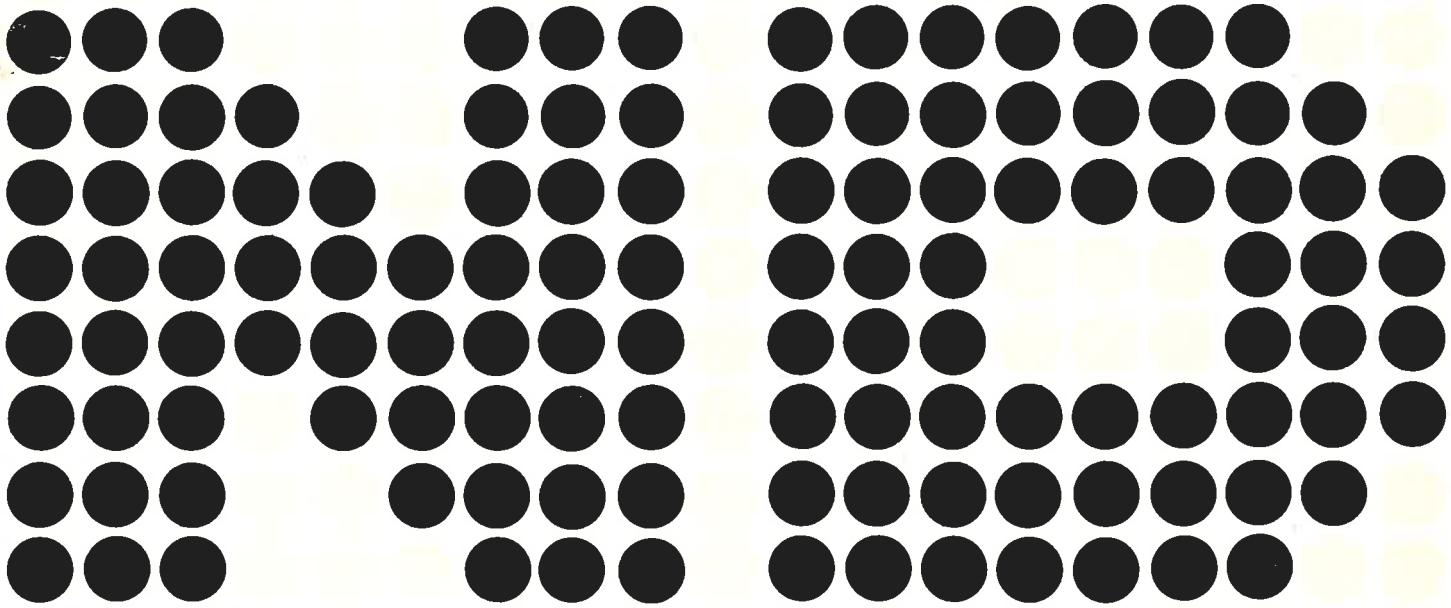
NORD STANDARD FORTRAN
REFERENCE MANUAL

A/S NORSK DATA-ELEKTRONIKK



NORD STANDARD FORTRAN
REFERENCE MANUAL

REVISION RECORD

Revision	Notes
10/74	Fourth version

ND-60.011.04

October 1974

ABSTRACT

This NORD STANDARD FORTRAN Reference Manual is written for programmers using the NORD STANDARD FORTRAN System.

The manual assumes a basic knowledge of the FORTRAN language. However, extensive use of examples throughout the manual should be of help to clarify some of the difficulties.

The manual contains the information required to produce and run a FORTRAN job.

--ooOoo--

TABLE OF CONTENTS

+++
+

Chapters: Page:

Appendices:

1 INTRODUCTION

The NORD STANDARD FORTRAN System provides a convenient language for expressing mathematical and scientific problems in a familiar notation.

A set of FORTRAN statements, presented as a source program to the FORTRAN compiler, produces an object program that contains the machine language instructions for solving a problem. Compilation is carried out sequentially, from one subprogram to the next; each subprogram is independently compiled. Once a program is compiled, and if no errors are detected by the compiler, a program may be repeatedly loaded by the loader and executed on the NORD computer with varying sets of data.

The NORD STANDARD FORTRAN System is compatible with standard FORTRAN (ref. USA STANDARD FORTRAN, USAS X3.9.1966), except the deviations quoted in Appendix I, and source programs written in the NORD STANDARD FORTRAN language will possibly require a few minor modifications to be accepted by the FORTRAN compiler on other, larger computer systems.

1.1 Minimum Machine Configuration

In order to implement the NORD STANDARD FORTRAN System on a NORD-1 computer, the minimum hardware configuration required is:

- 1 Basic NORD-1 computer with minimum 16K memory unit, or
  1 Basic NORD-10 computer with minimum 16K memory unit.
- Floating point hardware arithmetic unit.
- Input device : Teletype, paper tape reader or card reader.
- Output device: Teletype, paper tape punch or line printer.

2 ELEMENTS OF NORD STANDARD FORTRAN

2.1 Constants

Eight basic types of constants are used in the NORD STANDARD FORTRAN: Integer, Double integer, Real, Double precision real, Complex, Logical, Octal, and Hollerith. The type of a constant is determined by its form. The computer word structure for each type is given in Appendix E.

2.1.1 Integer

An integer constant consists of up to five decimal digits in the range of $-2^{15} \leq n \leq 2^{15}-1$. An integer constant occupies one word of NORD main storage.

Examples:

63	-3241	896
247	27963	-4343

2.1.2 Double Integer

A double integer constant consists of up to 10 digits in the range of $-2^{31}=-2147483648 \leq n \leq 2147483647=2^{31}-1$. A double integer constant occupies two consecutive storage locations.

Examples:

-444444	999000000

2.1.3 Real

Real constants are represented by a string of up to ten digits. A real constant may be expressed with a decimal point or with a fraction and an exponent representing a power of ten. The forms of real constants are:

.nE	.nE$^{\pm}$s	n.	n.E$^{\pm}$s
n.n	n.nE$^{\pm}$s	.n	

n is the base; s is the exponent to the base 10. The plus sign may be omitted for a positive s. The range of s is 0 through 99.

If the range of a real constant is exceeded, the constant is set to the maximum value, and a diagnostic is provided.

A real constant occupies three consecutive main storage locations.

Examples:

3.1415768	-314.	.013469
.31416E1	3.14E06	-31.415E-1
-0.31415E+01		

2.1.4 Double Precision Real

Double precision constants may be expressed by one to 23 significant decimal digits. Their forms are much alike to real constants, but a D corresponds to E in the exponent part. The range is also equivalent to that of reals.

A double precision constant occupies six consecutive main storage locations.

Examples:

```
0.0D0          -1340.D3          3.1415926535D+1
+8.5D-2              .4D04
```

2.1.5 Complex

Complex constants are represented by pairs of real constants separated by a comma and enclosed in parentheses

$$(R_1, R_2)$$

R_1 represents the real part of the complex number, and R_2 the imaginary part. Either constant may be preceded by a - sign.

If the range of the reals comprising the constant is exceeded, a compiler diagnostic is provided. Diagnostics also occur when the pair of numbers consists of integer constants, including (0,0).

A complex constant occupies six main storage locations.

Examples:

NORD FORTRAN IV Representation:	Complex Numbers:
(1.,3.80)	1.+3.80i
(8.1,16.2)	8.1+16.2i
(-11.09,1.2E-3)	-11.09+0.0012i
(1.,0.)	1
(0.,-1.)	-i

2.1.6 Logical

Logical constants are represented by one of the following notations:

.TRUE.

.FALSE.

A logical constant occupies one main storage location; the system represents .TRUE. by 1 and .FALSE. by 0.

2.1.7 Octal

An octal constant is denoted by one to six octal digits postfixed by the letter B. If more than six digits are specified, the last six are significant only.

Examples:

123456B -7B 177777B

Note: If six digits are specified, the most significant one should be a 0 or a 1, only, which are represented by one bit.

2.1.8 Hollerith

A Hollerith constant is a string of alphanumeric characters of the form nHf or 'f'; n is an unsigned decimal integer less than 81 representing the length of the field f. Spaces are significant in the field f. When n is not a multiple of 2, the last computer word is left justified with ASCII space filling the remainder of the word. Hollerith constants may not be used within expressions.

Example:

2 H0K	3 HSUM	'FORTRAN␣IV'
2 H␣A	6 HEXAMPL	12 HCOMPLEX␣DATA
2 HA6	6 HNORD-1	9 HHOLLERITH
1 H1	4 HDATA	

2.2 Variables

Variable names are alphanumeric identifiers that represent specific storage locations.

The NORD STANDARD FORTRAN compiler recognizes simple and subscripted variable names.

2.2.1 Simple Variables

The type of the variable may be defined in a TYPE declaration (Chapter 4). Otherwise, the type is determined by the first letter of the variable name. The initial characters I, J, K, L, M and N indicate integer variables; other initial letters indicate real variables.

A simple variable represents a single quantity; a subscripted variable represents either an array or one element within an array. A symbolic name consists of one to six alphanumeric characters, the first of which must be alphabetic.

Examples of simple integer variables:

N	LIN	I12
K2P11	NODE	JTEST

Examples of simple real variables:

VECT	B2306	OLE	SIXSIX
PET26	A1B	ATE	

2.2.2 Subscripted Variables

A subscripted variable is represented by an alphanumeric identifier followed by a one, two, three or four dimensional subscript enclosed in parentheses. If the subscript has more than four dimensions, a diagnostic is issued. The identifier is the name of the array; subscripts may be constants, variables, or expressions with integer values. A non-integer value will cause a compiler diagnostic.

A subscripted variable references a single element in an array, the subscript describes the relative location of the element within the array.

Subscript Forms

A subscript dimension may have the form of any integer expression. Mixed mode or real expressions are not allowed.

Examples:

Legal:	Illegal:
ARRAY(10 * NUM + 5, 20)	ARRAY(10. * NUM + 2, 20.)
A(I(J))	A(A(2.))
B(I + 2, J + 3, K + 4)	B(X + 2., Y + 3., Z + 4.)
C(IABS(J))	C(ABS(X))

2.2.3 Arrays

An array is a block of successive memory locations for storage of variables. In certain contexts, the entire array may be referred to by the array name without subscripts. Each element of an array is referenced separately by the array name plus the subscript notation. Arrays may have one, two, three or four dimensions.

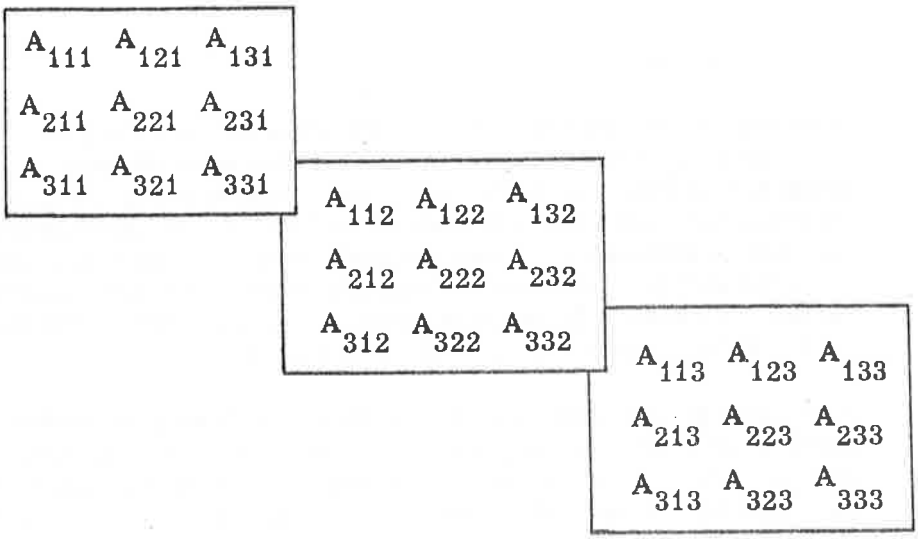
The array name and the dimensions of the array must be declared at the beginning of the program in a DIMENSION, COMMON or a type statement. The type of array is determined by the array name or the type declaration. The number of dimensions in an array subscript indicates the dimension of the array; the magnitude of each dimension indicates the maximum value that the subscript may take. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array.

The amount of memory allocated to an array depends on the array type and dimensions.

The compiler does not necessarily assign sequential storage to two or more arrays.

2.2.3.1 Array Structure

Elements of arrays are stored by columns in ascending order of storage location. The ordering of elements in an array follows the rule that the first subscript varies most rapidly and the last subscript varies least rapidly. In the array declared as A(3,3,3):



The planes are stored in order, starting with the first, as follows:

$A_{111} \rightarrow L$ $A_{121} \rightarrow L+9 \ldots A_{133} \rightarrow L+72$,

$A_{211} \rightarrow L+3$ $A_{221} \rightarrow L+12 \ldots A_{233} \rightarrow L+75$,

$A_{311} \rightarrow L+6$ $A_{321} \rightarrow L+15 \ldots A_{333} \rightarrow L+78$,

since one element in K occupies 3 locations.

The location of an array element with respect to the first element is a function of the maximum array dimensions and the type of the array. Given DIMENSION A(L,M,N) the location of A(i,j,k) with respect to the first element of the array A, is given by:

$$A + \left[i - 1 + L\,(j - 1 + M\,(k - 1)) \right] * E$$

The quantity in brackets is the subscript expression. It must be a positive integer value. E is the element length in terms of the number of computer words needed for each element of the array.

For real arrays, E = 3; for integer arrays E = 1.

2.2.3.2 Array Notation

A subscript describing an array notation cannot have more dimensions than are specified for the array; thus the elements of the one-dimensional array $A(ID_1)$ may not be referred to as A(I, J, K, L), A(I, J, K) or A(I, J). A diagnostic will be given if this is attempted. However, any two-, three-, or four-dimensional array may always be referred as if it were a one-dimensional.

The array name without a subscript references the entire array when it is used in an I/O list, a specification statement other than DIMENSION, or as a parameter of a function or subroutine subprogram.

2.3 Statements

Statements are the basic functional units of the FORTRAN language. An executable statement performs a calculation or directs control of the program; a non-executable statement provides the compiler with information regarding variable structure, array allocation, storage sharing requirements. Assignment, control, and input/output statements are executable. The non-executable statements are specification statements, function defining statements, and the DATA, FORMAT, PROGRAM, FUNCTION, and SUBROUTINE statements.

A statement consists of an initial line which may be followed by any number of continuation lines. The characters of a statement are written, one per column, in columns 7 through 72. Continuation lines are marked by a non-blank character (other than blank or zero) in columnt 6. No more than one statement may be written on a line.

A unique label may be assigned to any statement and must be assigned to one referred to by other statements. A statement label is a numeric string of one to five digits; leading zeroes are ignored. Thus 0012 is equivalent to 12 or 012 when used as a statement label. The label may be placed anywhere in the label field. Trailing spaces are ignored. Thus ␣␣12␣, 12␣␣␣and␣␣␣12 all refer to the same label.

2.4 Program Units

A NORD STANDARD FORTRAN program consists of one main program and, optionally, one or more subprograms. The term program unit refers to either the main program or a subprogram.

A main program is a set of statements and comments forming a self-contained computing procedure; it must contain at least one executable statement. A PROGRAM statement may be used as the first statement of a main program, but is not necessary. A main program may not contain a FUNCTION, or a SUBROUTINE statement.

A subprogram is also a set of statements and comments. A procedure subprogram contains at least one executable statement and is headed by either a FUNCTION or SUBROUTINE statement.

All program units must be terminated by an END statement.

3 EXPRESSIONS AND REPLACEMENT STATEMENTS

3.1 Arithmetic Expressions

An arithmetic expression is a constant, variable (simple or subscripted), an evaluated function, or any combination of these separated by arithmetic operators, commas, or parentheses to form a meaningful mathematical expression.

Arithmetic Operators:

+	addition	-	subtraction	*	multiplication
/	division	**	exponentiation		

3.1.1 Elements

The elements of arithmetic expressions are formed as follows:

A primary is an arithmetic expression in parentheses, a constant (positive or zero), variable, array element, or function reference:

(A+B)	(-A*B)	((A**B)-(A*B))
124	12.4E-2	0
X	A(I,J)	SIN(V)

A factor is a primary, or a primary**a primary:

(A+B)	(A+B)**X	I**2

A term is a factor, a term/factor, or a term*term:

A**B	(A**B)/X	((A**B)/X)*SIN(V)

A signed term is immediately preceded by a plus or minus:

-A**B	-X	-(-A*B)

A simple arithmetic expression is a term, or two simple arithmetic expressions separated by plus or minus:

(A+B)+X	X/2.314	Y/SIN(X)-A**B

An arithmetic expression is a simple arithmetic expression, or a signed term plus or minus a simple arithmetic expression:

-X/Y	I**2+K	-A**B-X/Y

3.1.2 Rules for Forming Expressions

Two arithmetic operators may not be adjacent to each other; X + - Y is an illegal expression. The subtraction operator may not be used as a sign of negation. -X implies 0-X and must be enclosed in parentheses when preceded by another operator: X + (-Y) is a legal expression.

Parentheses may be used to indicate grouping as in ordinary mathematical notation, but they may not be used to indicate multiplication: (X) (Y) does not imply (X)* (Y); nor does juxtaposition imply multiplication: XY does not imply X* Y.

Any primary may be raised to a power that is a positive or negative integer primary, but only a positive real primary can be raised to a real power. Real and integer quantities may be mixed in the same expression.

A negative primary may not be raised to a power that is a real number: (-15.0)** 2.5 is illegal. A primary with a zero value may not be raised to a power value as zero. An element may not be evaluated if its value is not mathematically defined. Diagnostics are given under run time.

3.1.3 Order of Evaluation

When the hierarchy of operations in an expression is not completely specified by parentheses, the operations are performed in the following order:

**	exponentiation	performed first
/	division	performed next
*	multiplication	
+	addition	performed last
-	subtraction	

Within a sequence of consecutive multiplications and/or divisions, or additions and/or subtractions, when the order is not explicitly indicated by parentheses, expressions are evaluated from left to right.

Whenever ambiguity is possible in the evaluation of an expression, parentheses should be used. The ambiguous expression A**B**C can be clarified as (A**B)** C or A**(B**C) only by parentheses.

Examples:

Valid Expressions	Invalid Expressions
A*(-B)	A*-B
A** (B**C)	
(A** B)** C	** B**C
-B+C	I**A
A-B+C	(-A)** C
A-(B+C)	
-(A+B)** C evaluated as -((A+B)** C)	
-(A+B)	
J** I	
A** I	

3.2 Mixed Mode Arithmetic Expressions

Arithmetic expressions can contain mixed types of constants and variables. Mixed mode arithmetic is accomplished through the special library conversion subroutines (Appendix D).

The order of dominance of the operand types within an expression is complex-double precision-real-double integer-integer.

In mixed mode arithmetic, the mode used to evaluate any portion of an expression is determined by the dominant type so far encountered within the expression, and the normal hierarchy of arithmetic operations; integer mode will be used when an integer type is first encountered and will be converted to real mode when a real type is encountered.

The following table indicates how the mode is determined from the possible combinations of variables.

+ - * /	Integer	Double integer	Real	Double precision	Complex
Integer	Integer	Double integer	Real	Double precision	Complex
Double integer	Double integer	Double integer	Real	Double precision	Complex
Real	Real	Real	Real	Double precision	Complex
Double precision	Double precision	Double precision	Double precision	Double precision	Complex
Complex	Complex	Complex	Complex	Complex	Complex

Examples:

1) Given A,B type real; I,J type integer. The mode of evaluating the expression (A*B-I+J) will be real because the dominant operand is type real. It is evaluated:

$A*B \rightarrow R_1$ real

Convert I to real

$R_1 - I \rightarrow R_2$ real

Convert J to real

$R_2 + J \rightarrow R_3$ real

2) The use of parentheses can change the evaluation. A, B, I, J are defined as above. (A*B-(I-J)) is evaluated:

$A*B \rightarrow R_1$ real

$I-J \rightarrow R_2$ integer

Convert R_2 to real

$R_1 - R_2 \rightarrow R_3$ real

3) The order of the elements in an expression can change the evaluation. A, B, I, J are defined as above. The expression (J-I+A+B) is evaluated:

$J-I \rightarrow R_1$ integer

Convert R_1 to real

$R_1 + A \rightarrow R_2$ real

$R_2 + B \rightarrow R_3$ real

Rules:

1) The order of dominance of the standard operand types within an expression from highest to lowest is

COMPLEX
DOUBLE PRECISION
REAL
DOUBLE INTEGER
INTEGER

2) The mode of an evaluated arithmetic expression is referred to by the name of the dominant operand type.

3) In expressions of the form A**B the following rules apply:

- B may be negative when the form is A**(-B).
- For the standard types the mode/type relationships are:

Type A \ Type B	Integer	Double integer	Real	Double precision	Complex
Integer	Integer		Real		
Double integer	Double integer				
Real	Real		Real		
Double precision	Double precision				
Complex	Complex				

Mode of A**B

The empty squares denote illegal expressions.

3.3 Arithmetic Replacement Statement

The general form of the arithmetic replacement statement is

v = e

e is an arithmetic ecpression and v is any variable name, simple or subscripted written without a sign. The operator = means that v is replaced by the value of expression e, with conversion for mode if necessary.

Examples:

```
REST      = X + Y * A
SUM       = X + SIN(X)
ARG(I,J)  = X + 2. * Y(I + 1)
PER(1)    = 5.2 + X ** Y
```

3.4 Mixed Mode Replacement Statement

Although the type of an evaluated expression is determined by the type of the dominant operand, this does not restrict the types that the identifier v may assume.

Arithmetic Replacement Statement

v = e

v is an identifier, e is the evaluated arithmetic expression.

Rules for Assignment for e to v

v type	e type	Assignment
Integer	Integer	Assign
Integer	Double integer	Convert double integer to integer and assign
Integer	Real	Fix and assign
Integer	Double prec.	Double precision fix and assign
Integer	Complex	Fix real part and assign
Double integer	Integer	Convert integer to double integer and assign
Double integer	Double integer	Assign
Double integer	Real	Fix to double integer and assign
Double integer	Double prec.	Double precision fix to double integer and assign
Double integer	Complex	Fix real part to double integer and assign
Real	Integer	Float and assign
Real	Double integer	Float and assign
Real	Real	Assign
Real	Double prec.	Double precision evaluate and real assign
Real	Complex	Assign, real part of e
Double prec.	Integer	Double precision float and assign
Double prec.	Double integer	Double precision float and assign
Double prec.	Real	Real evaluate, Double precision assign
Double prec.	Double prec.	Assign
Double prec.	Complex	Real part evaluate, Double precision assign
Complex	Integer	Float (e) → real part, 0 → imaginary part
Complex	Double integer	Float (e) → real part, 0 → imaginary part
Complex	Real	e → real part, 0 → imaginary part
Complex	Double prec.	Real converted e → real part, 0 → imaginary part
Complex	Complex	Assign

Examples:

1) A = I+J is evaluated as:

I+J → R_1 integer

Convert R_1 to real

Store R_1 in A

2) I = J+A is evaluated as:

Convert J to real

J+A → R_1 real

Convert R_1 to integer

Store R_1 in I

3.5 Logical Expressions

A logical expression has the general form

$$0_1 \text{ op } 0_2 \text{ op } 0_3 \ldots .$$

The forms 0_i are logical variables or relational expressions; and op is either the logical operator .AND. indicating conjunction, or .OR. indicating disjunction.

The logical operator .NOT. indicating negation appears in the form

$$.\text{NOT. } 0_1$$

The value of a logical expression is either true or false. Logical expressions are generally used in logical IF statements. (See Section 5.3).

Rules:

1) The hierarchy of logical operations is:

First	.NOT.
then	.AND.
then	.OR.

2) A logical variable or a relational expression is, in itself, a logical expression. If L_1 and L_2 are logical expressions, then

$.\text{NOT. } L_1$

$L_1 .\text{AND. } L_2$

$L_1 .\text{OR. } L_2$

are logical expressions. If L is a logical expression, then (L) and ((L)) are logical expressions.

3) If L_1 and L_2 are logical expressions and op is .AND. or .OR. then L_1 op op L_2 is always illegal.

4) The logical operator .NOT. may appear in combination with .AND. or .OR. only as follows:

.AND..NOT.

.OR..NOT.

.AND.(.NOT. ...)

.OR.(.NOT. ...)

.NOT. may appear with itself only in the form

.NOT.(.NOT.(.NOT. .....

Other combinations will cause compiler diagnostics.

5) If L_1 and L_2 are logical expressions, the logical operators are defined as follows:

.NOT. L_1	is false only if L_1 is true
L_1 .AND. L_2	is true only if L_1 and L_2 are both true
L_1 .OR. L_2	is false only if L_1 and L_2 are both false

Examples of logical expressions:

Valid expressions:	Illegal expressions:
A.OR.B	A.NOT..OR.B
A.AND.B	A.OR..NOT..NOT.B
A.OR.B.AND.C.OR.D	X.GT.B.AND.C
.NOT.A.AND.B.AND.C	
.NOT.(A.AND.B)	
X.GT.Y.AND.A	
A.AND..NOT.B	

A, B and C are logical variables, X and Y are real.

3.6 Relational Expression

A relational expression has the form:

$$q_1 \text{ op } q_2$$

where q_1 and q_2 are arithmetic expressions; op is an operator belonging to the following set:

Operator:	Meaning:
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

A relation is true if q_1 and q_2 statisfy the relation specified by op.

A relation is false if q_1 and q_2 do not satisfy the relation specified by op.

Rules:

1) Use a relational operator between two arithmetic expressions:

q_1 op q_2

2) It is not permissible to use the form

q_1 op q_2 op q_3

Instead separate two relational expressions with a logical operator .AND. or .OR. in any of the form

q_1 op q_2 .AND. q_3 op q_4

q_1 op q_2 .OR. q_3 op q_4

3) The evaluation of a relation of the form q_1 op q_2 is from left to right.

The relations q_1 op q_2, q_1 op (q_2), (q_1) op q_2 and (q_1) op (q_2) are equivalent.

Examples:

```
A.GT.5.2
RX-X(5)*A.LT.Y
B-C.EQ..5
X(I).GE.X(I-1)
I.LE.10
```

3.7 Logical Replacement Statement

The general form of a logical replacement statement is

E = L

where E is a variable of type logical and L may be a logical or relational expression, or any of the logical values .TRUE. or FALSE.

Examples:

```
LOGICAL  L1, L2, L3, L4

L1 = .TRUE.
L2 = .FALSE.
L3 = L1.OR.L2
L4 = L1.AND..NOT.L3
L1 = X.GE.Y
L2 = L1.OR.Y.EQ.2.
```

Note: It is illegal to assign a logical or relational expression to an arithmetic variable, or to assign an arithmetic expression to a logical variable.

4 TYPE DECLARATIONS AND STORAGE ALLOCATIONS

Statements of this kind are also called declarative statements. Declarative statements are non-executable statements that:

- assign word structure to variables (TYPE),
- reserve storage for arrays and single variables (DIMENSION, COMMON),
- designate shared storage (COMMON, EQUIVALENCE), and
- assign initial values to variables (DATA).

4.1 TYPE Statement

The TYPE statement provides the compiler with information about the structure of variable or function identifiers. It overrides or confirms the type implied by the first character of the identifier, and it may provide dimension information. The TYPE statement has the following form:

$$t\ v_1, \ldots, v_n$$

t is INTEGER, DOUBLE INTEGER, REAL, COMPLEX, DOUBLE PRECISION or LOGICAL, and the v_i are variable name, array name, function name, or array declarator.

Example:

INTEGER	A, XI1, I1, HEP, D36F	(1 word/element)
DOUBLE INTEGER	IDOUBL, DWORD (10)	(2 words/element)
REAL	INTER, ITEST, K25, ALFA	(3 words/element)
DOUBLE PRECISION	DP	(6 words/element)
COMPLEX	C1	(6 words/element)
LOGICAL	L1, L2, X, Y(5)	(1 word/element)

Rules:

1) The TYPE declaration is non-executable and must precede the first executable statement in a given program.

2) If an identifier is declared in two or more TYPE declarations a compiler diagnostic will occur.

3) An identifier not decleared in a TYPE statement will be an integer if the first letter of the identifier is I, J, K, L, M, N; for any other letter it will be real.

4) An array identifier in the list designates the entire array.

4.2 DIMENSION Statement

Storage may be reserved for arrays with non-executable statements, DIMENSION, COMMON and type.

DIMENSION v_1 (i_1), . . . , $v_n(i_n)$

Each v(i) is an array declarator. v_i are the array names; (i_1) are subscripts containing 1, 2, 3 or 4 integer constant subscript dimensions separated by commas. The number of dimensions indicates the dimensions of the array. The magnitude of the value given for each dimension indicates the maximum value that the dimension may take in any subsequent reference.

From information in a DIMENSION statement, the compiler determines the number of computer words to reserve for the array named in the statement.

In the following statement, the number of elements in the array is 125; the array has three dimensions and its elements are real numbers.

```
DIMENSION SPACE (5,5,5)
REAL SPACE
```

The value of a subscript dimension may never be less than 1.

The number of computer words reserved for the array, SPACE, is 375. This is three times the number of elements in the array because the type of the array is REAL, and in the NORD-1 computer, a real number uses three computer words or 48 bits.

An integer uses one computer word, 16 bits. Therefore, in the following example the number of computer words reserved for the array ISP is 125.

Example:

```
DIMENSION ISP (5,5,5)
DIMENSION A(30), I22(10,2), AB(6,20)
DIMENSION H(5,5)
COMPLEX  H
```

The number of elements in H is 25. 6 words are used to form a complex element; therefore, the number of memory locations reserved for H is 150.

4.2.1 Adjustable Dimensions

In a subprogram (see Chapter 6), a formal argument may be declared to be an array in a type or DIMENSION statement. The corresponding actual argument is an array name. The dimensions of the formal argument may be transmitted as arguments, or they may be constants of the subprogram. For example,

```
SUBROUTINE   SUB(A,I)
DIMENSION    A(J,5,5)
```

The number and values of the dimensions need not be the same in both the calling and the called routines. Storage for the array is not allocated in the subprogram and the dimension information is used only to compute addresses. The product of the maximum dimensions of the formal argument must not exceed the main storage assigned to the actual argument.

4.3 COMMON Statement

A program may be divided into independently compiled subprograms that use the same data. The COMMON statement reserves storage areas - blank or labeled - that can be referenced by more than one subprogram.

$$\text{COMMON}/x_1/a_1 \; .. \; /x_n/a_n$$

x_i are alphanumeric identifiers, and each a_i is a list composed of simple variable identifiers and array identifiers, subscripted or non-subscripted.

A list a_i may not contain formal parameters. If a non-subscripted array name appears, the dimensions must be defined by a DIMENSION statement in that program unit. Arrays may be dimensioned in the COMMON statement by a subscript string following the array identifier. If an array is dimensioned in both a COMMON statement and a DIMENSION statement, a compiler diagnostic results.

An identifier x_i may be a name of one to five alphanumeric characters or blank. A non-blank name identifies the storage as labeled common; a blank name identifies blank common. If the name is blank, the first two slashes may be omitted. Only one name may be assigned to labeled common, but the name may be specified more than once.

All labeled common storage areas are assigned together in the order of appearance regardless of the number of identifiers; all blank common storage areas are assigned together in the order of appearance.

Examples:

```
COMMON A, B, C
COMMON // A, B, C, D
COMMON /BLOK/ A, B(10) /BLOK2/ C(10), D(10,10)
COMMON /ABC/  D(15), ABC, PER, I1(50)
```

4.4 Common Blocks

The COMMON statement provides the programmer with a means of reserving blocks of storage areas that can be referenced by more than one subprogram. The statement reserves both blank and labeled blocks.

If a subprogram does not use all of the locations reserved in a common block, unused variables may be necessary in the COMMON statement to ensure proper correspondence of common areas:

```
Main program : COMMON/SUM/A,B,C
Subprogram   : COMMON/SUM/E,F,G
```

In the above example only the variables E and G are used in the subprogram. The unused variable F is necessary to space over the area reserved by B.

Rules:

1) COMMON is non-executable and must precede the first executable statement in the program. Any number of COMMON statements may appear in a program unit.

2) Labeled common block identifiers are used only for block identification within the compiler; they may be used elsewhere in the program as other kinds of identifiers.

3) An identifier in one common block may not appear in another common block. If it does, the identifier is doubly defined and an error message will result.

4) The order of the arrays in a common block is determined by the COMMON statement.

5) At the beginning of program execution, the contents of the common block are undefined unless specified by a DATA statement.

The length of a common block in computer words is determined from the number and type of the list identifiers. In the following statement, the length of the common block A is 26 computer words. The origin of the common block is Q(1), (Q and R are real, NR is integer).

Examples:

1) Labeled common:

COMMON/A/ Q(4), R(4), NR(2)

origin		
	Q	(1)
	Q	(2)
	Q	(3)
	Q	(4)
+12	R	(1)
	R	(2)
	R	(3)
	R	(4)
+24	NR	(1)
	NR	(2)

Each real variable requires three computer words

2) Blank common:

COMMON A, B(2), K

COMMON N(2), M(2)

origin			
	A		
	A		Real
	A		
	B	(1)	
	B	(1)	Real
	B	(1)	
	B	(2)	
	B	(2)	Real
	B	(2)	
	K		
	N	(1)	
	N	(2)	
	M	(1)	
	M	(2)	

3) Rearrangement of common:

Main program:

COMMON /EX/ TEMP(20)

The labeled common, EX, occupies 60 storage locations.

Subprogram:

COMMON /EX/ B(10), I(10), J(20)

Thelabeled common occupies the same 60 storage locations as in the main program, however, 30 locations are used by the real array B, 10 locations are used by the integer array I and 20 locations are used by the integer array J.

4.5 EQUIVALENCE Statement

The EQUIVALENCE statement permits storage to be shared by two or more variables. It does not equate these variables mathematically.

EQUIVALENCE $(k_1), \ldots, (k_n)$

Each k_i is an equivalence group of two or more variables or array elements separated by commas: $a_1, a_2, \ldots, a_m$. If an element a_i has a subscript, the subscript must contain only constants. No formal parameters may appear in an EQUIVALENCE statement. Every element a_i in one equivalence group is assigned the same storage. If a real number is assigned the same storage as an integer, only the first word of the real number is shared with the one-word integer.

The first elements of arrays may be aligned by equivalencing the array names; elements of integer, logical, real, and complex arrays may be aligned by equivalencing subscripted variables (the subscripts must be integer constants). Array lengths need not be equal.

Example:

If two arrays, not in common, are equivalenced

```
DIMENSION A(3), B(2), C(4)
INTEGER A,B,C
EQUIVALENCE (A(3), C(2))
```

storage locations are assigned as follows:

L	A(1)	
L+1	A(2)	C(1)
L+2	A(3)	C(2)
L+3		C(3)
L+4		C(4)
.		
.		
.		
M	B(1)	
M+1	B(2)	

However, if two arrays in common are equivalenced

```
DIMENSION C(4)
COMMON A(3), B(2)
EQUIVALENCE (B(2), C(2)
```

storage locations are assigned as follows:

L	A(1)	
L+1	A(2)	
L+2	A(3)	
L+3	B(1)	C(1)
L+4	B(2)	C(2)
L+5		C(3)
L+6		C(4)

The EQUIVALENCE statement does not rearrange common, but arrays may be defined as equivalent so that the length of a common block is changed. The origin of the common block may not be changed by an EQUIVALENCE statement.

Rules:

1) EQUIVALENCE is non-executable and must precede the first executable statement in the program or subprogram.

2) The EQUIVALENCE statement must follow after DIMENSION or COMMON.

3) No more than one element in an EQUIVALENCE set may belong to COMMON.

4) An identifier used as a formal parameter cannot also be used in an EQUIVALENCE statement.

5) EQUIVALENCE cannot rearrange COMMON, however, arrays may be equivalent so that they change the length of the common block.

6) An identifier may appear more than once in an EQUIVALENCE statement.

7) An identifier in a COMMON statement used in an EQUIVALENCE set is the base identifier for the EQUIVALENCE statement. When none in the set belongs to COMMON, the identifier with the lowest address becomes the base identifier. All other elements in the set are referenced to the base identifier.

Example:

Align elements of two arrays:

```
      DIMENSION A(10,5), I(150)
      EQUIVALENCE (A,I)
    5 READ (N,100) A
      :
   10 READ (N,110) I
      :
```

The EQUIVALENCE statement assigns the first element of array A and array I to the same storage location. The READ statement 5 stores array A in consecutive locations. Before statement 10 is executed all operations using A should be completed as the values of array I will be read into the storage locations previously occupied by A.

It should be noted that I(1), I(2), and I(3) are stored into the three consecutive locations making up A(1).

Example:

```
EQUIVALENCE (A,B),(C,D),(E,F),(A,F),(B,D)
```

This statement will be interpreted as and identical to the following statement:

```
EQUIVALENCE (A,B,C,D,E,F)
```

4.6 DATA Statement

The DATA statement assigns constant values to variables or arrays in the source program. It may be used by itself or with a DIMENSION statement.

$$\text{DATA } k_1/d_1/,\ldots,k_n/d_n/$$

k_i are lists containing the names of variables or array elements; and d_i are corresponding lists of constants (signed or unsigned).

Multiple entries in a list are separated by commas. There must be a one-to-one correspondence between the elements of a list k_i and a list d_i. This correspondence establishes the initial values of the elements of list k_i.

When an element of a list k_i is an array element, the subscript must contain only integer constants. An element of a list k_i may not appear as a formal parameter.

Examples:

1)
```
DIMENSION GRADE (8)
REAL GRADE
INTEGER I
DATA GRADE(1), GRADE(2), GRADE(3), GRADE(4), GRADE(5)
     GRADE(6), /60.,65.,70.,75.,80.,85.,/,I/1/
```

Some elements of the array GRADE are set to the initial values specified in the associated list: GRADE(1) is to contain the initial value 60., GRADE(2) the initial value 65., and so forth. In the same statement the integer variable I is set to the initial value 1.

Repetition factor:

```
DIMENSION A(10)
DATA A/1.0, 9* 2.0/
```

The value 2.0 will be put into nine consecutive elements.

```
DIMENSION A(10)
DATA A/1., 2., 5., 2.5, 0.5, 3., 10., 20., 10., 1.0/

COMPLEX CX
DATA CX/(1.0, 2.0)/

LOGICAL L1(2)
DATA L1/.TRUE., .FALSE./

DIMENSION OUT(3)
DATA OUT/4HTHIS, 3H␣IS, 4HTRUE/
```

4.7 BLOCK DATA Statement

This is of the form:

```
BLOCK DATA
```

and may only appear as the first statement of a block data subprogram. Such subprograms are used to enter initial values into elements of blank and labeled common blocks. Only type statements, EQUIVALENCE, DATA, DIMENSION, and COMMON statements are permitted in a block data subprogram.

If any entity of a given common block is being given an initial value in such a subprogram, a complete set of specification statements for the entire block must be included, even though some of the elements of the block do not appear in DATA statements.

The block data subprogram should precede all the executable program units.

Example of a block data subprogram:

```
BLOCK DATA
DIMENSION ARR(5)
INTEGER AA(10)
COMMON /BLOC1/ARR,/BLOC2/AA
DATA ARR/5*1.0/, AA(1)/1/
END
```

5 CONTROL STATEMENTS

Program execution normally proceeds from statement to statement as they appear in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section. Control may be transferred to an executable statement only; a transfer to a non-executable statement will result in a program error which is usually recognized during compilation. With the DO statement, a predetermined sequence of instructions can be repeated any number of times by stepping a simple integer variable after each iteration.

5.1 Statement Identifiers

Statements are identified by unsigned numbers, 1 to 32767, which can be referred to from other sections of the program. An identifier may occupy any of the first five columns of the coding form; blanks are squeezed out and leading zeroes are ignored, 1,01,001,0001 are identical. Such an identifying number is called a statement label.

5.2 GO TO Statements

GO TO statements provide transfer of control.

5.2.1 Unconditional GO TO Statement

GO TO k

This statement causes an unconditional transfer to the statement labeled k.

5.2.2 ASSIGN Statement

This statement has the form

ASSIGN k TO i

where k is a transfer label and i is an integer variable name. This statement is used in conjunction with assigned GOTO statements using the same integer variable.

Once having been mentioned in an ASSIGN statement, the integer variable should not be referred to in any statement other than an assigned GOTO statement. This applies until it has been redefined, since its content is an octal address after the execution of the ASSIGN statement.

5.2.3 Assigned GO TO Statement

The assigned GO TO statement has the form

$$\text{GO TO } i, (k_1, k_2, \ldots, k_n)$$

where i is an integer switch variable. Prior to the execution of an assigned GO TO statement, the variable i must have been given a label value by an ASSIGN statement. At run time, this label value is checked against the parenthesized list of labels. Then, if the actual label value coincides with any one of the list, a transfer is performed to the statement identified by this label. Otherwise, a run time error message will result, and the control is transferred to the statement of label k_1.

Example:

```
      ASSIGN  1 TO K
      .
      .
      .
   10 GO TO  K, (1,2,3)
      .
      .
      .
    1 ASSIGN  2 TO K
      GO TO  10
    2 K = 20
      OUTPUT (1) K
      .
      .
      .
```

5.2.4 Computed GO TO Statement

$$\text{GO TO } (k_1, \ldots, k_n), i$$

The k_i are statement labels; i is an integer variable.

Execution of this statement causes a branch to the statement identified by k_i, where i is the value of the integer variable at the time of execution. If i is less than 1 or greater than n, error message "RUN ERR GO" will result and control returns to label k_1.

Example:

```
INTEGER A,B,C
        A = 1
        C = 1

GO TO (10,20,30),C
        .
        .
        .
10 A = A + 2
GO TO (11,21,31),A        Control is transferred to the
                          statement labeled 31
```

5.3 IF Statements

Conditional transfer of control is provided by the arithmetic IF statement and the logical IF statement.

5.3.1 Arithmetic IF Statement

The arithmetic IF statement has three branches.

IF (e) k_1, k_2, k_3

e is an arithmetic expression and k_i are statement labels. This statement tests the evaluated quantity e and jumps to one of the labels k_1 according to the value of e.

$e < 0$	jump to k_1
$e = 0$	jump to k_2
$e > 0$	jump to k_3

Examples:

```
IF  (A*B-C*SIN(X)) 10,10,20
IF  (I) 5,6,7
IF  (A/B**2) 3,6,7
```

5.3.2 Logical IF Statement

IF (L) s

L is a logical or relational expression and s is a statement. If L is true (non-zero), the statement s is executed. If L is false (zero), continue in sequence to the statement following the logical IF.

Example:

```
IF (L) GO TO 10                          (L is logical)
IF (A.AND.B) X = SIN(Y)/P
IF (X.GE.2.) X = 2.
IF (Y.GT.5..OR.Y.LT.-5.) GO TO 100
```

5.4 DO Statements

The DO statement makes it possible to repeat a set of statements and to change the value of an integer variable during the repetition.

DO n i = m_1, m_2, m_3

DO n i = m_1, m_2

The DO loop begins with the DO statement and ends with the statement numbered n; i is the simple integer variable used as an index; m_i are the indexing parameters. m_1 is the initial value assigned to i; m_2 is the final value assigned to i. Each must be either an integer constant or an integer variable. m_3 is the increment added to i after each DO loop is executed. m_3 is an integer constant or an integer variable. If m_3 is omitted, it is assumed to have the value 1. m_1 and m_2 may be negative and m_3 must be greater than zero.

The statement label n which terminates the DO loop must be the number of an executable statement in the same program unit as the DO statement and must follow it. n may not be the label of any of the following:

GO TO statement

Arithmetic IF

RETURN

STOP

PAUSE

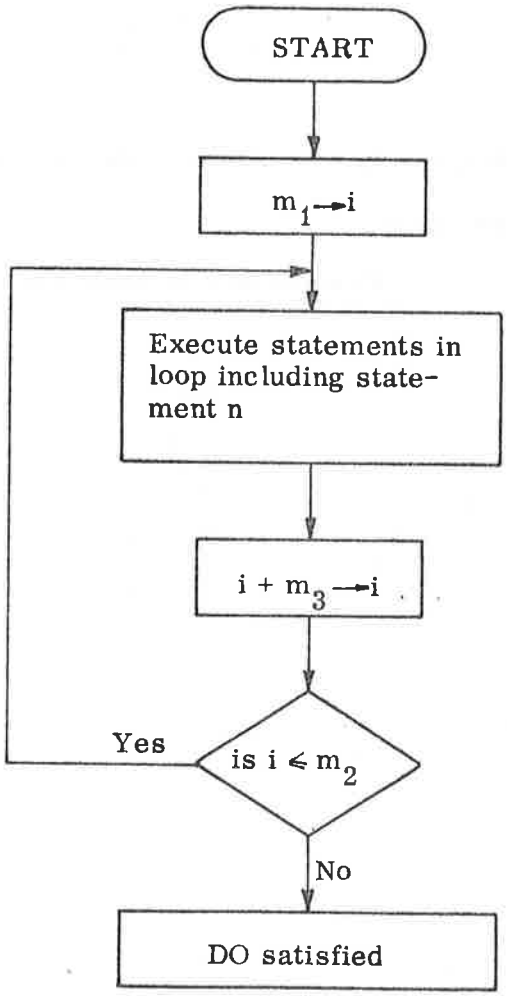
DO statement

5.4.1 DO Loop Execution

The DO statement, the statement labeled n, and any intermediate statements constitute a DO loop which consists of the following steps:

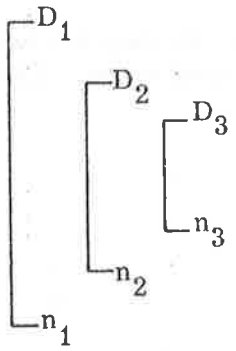
i is set to its initial value m_1 and the DO loop is executed. At the end of the DO loop i is increased by m_3 (or 1), and the value of i is compared with m_2. If i is less than or equal to m_2, the DO loop is executed. If i is greater than m_2, control passes to the statement immediately following n, and the DO loop is terminated.

Note that the DO loop is always executed at least once, even if m_1 exceeds m_2 on the initial entry. The following chart shows a DO loop.



5.4.2 DO Nests

A DO loop containing another DO loop is called a DO nest. The last statement of a nested DO loop must either be the same as the last statement of the outer DO loop or occur before it. If $D_1, D_2, \ldots, D_m$ represent DO statements, where the subscripts indicate that D_1 appears before D_2 appears before D_3, and $n_1, n_2, \ldots, n_m$ represent the corresponding limits of the D_i, then n_m must appear before $n_{m-1} \ldots n_2$ must appear before n_1.

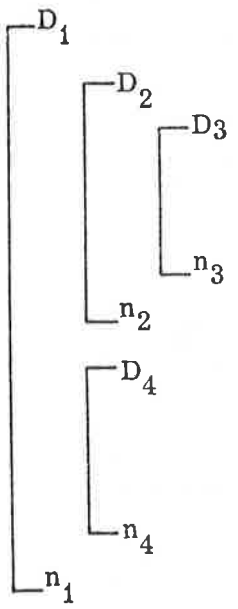


DO loops may be nested to the depth of ten at most.

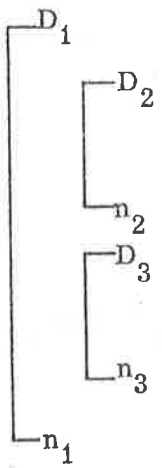
Examples:

DO loops may be nested in common with other DO loops:

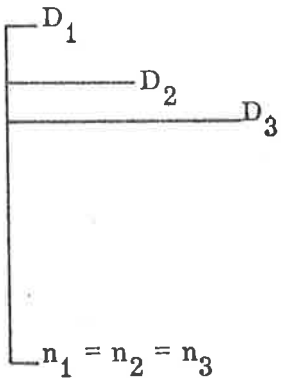
a)



b)



c)



a)
```
DO 1, I=1, 10, 2
.
.
.
DO 2 J=1, 5
.
.
.
DO 3 K=2, 8
.
.
.
3 CONTINUE
.
.
.
2 CONTINUE
.
.
.
DO 4 L=1, 3
.
.
.
4 CONTINUE
.
.
.
1 CONTINUE
```

b)
```
DO 100 L=2, LIMIT
.
.
.
DO 10 I=1, 10
.
.
.
10 CONTINUE
.
.
.
DO 20 K=K1, K2
.
.
.
20 CONTINUE
.
.
.
100 CONTINUE
```

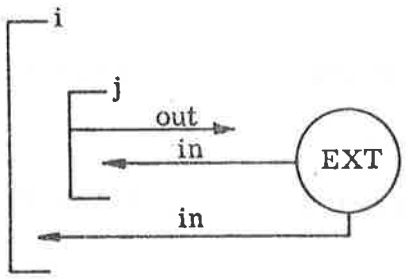
c)
```
DO 5 I=1, 5
DO 5 J=I, 10
DO 5 K=J, 15
.
.
.
5 CONTINUE
```

5.4.3 DO Loop Transfer

In a DO nest, a transfer may be made from one DO loop into a DO loop that contains it; and a transfer out of a DO nest is permissible.

The special case is transferring out of a nested DO loop and then transferring back to the nest. In a DO nest, if the range of i includes the range of j and a transfer out of the range of j occurs, then a transfer into the range of i or j is permissible.

In the following diagram, EXT represents a portion of the program outside of the DO nest.



If two or more DO loops terminate at the same statement and a transfer is made to the terminal statement outside the inner DO loop, the inner DO should have its own terminal statement.

No statement within the range of a DO may redefine or otherwise alter any of the indexing parameters of that DO.

Warning:

The compiler does not check for jumps from an external place to somewhere within the loop. If this is done, the result will depend on the last defined value of i.

5.5 CONTINUE Statement

CONTINUE

This statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If CONTINUE is used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement.

5.6 PAUSE Statement

PAUSE

PAUSE n

n is a positive, decimal number. When either statement is encountered, execution of the object program halts with PAUSE n or PAUSE output on the typewriter. By pressing an arbitrary character on the Teletype keyboard, program execution is continued with the statement immediately following PAUSE.

5.7 STOP Statement

STOP

STOP n

n is a positive, decimal number. When either statement is encountered, execution of the object program terminates. The program exits to the monitor system. STOP or STOP n is output on the typewriter.

In a main program the END statement will act as a STOP statement.

5.8 END Statement

END

END marks the physical end of a program unit. It is executable in the sense that it will effect return from a subprogram in the absence of a RETURN or a STOP in a main program.

6 PROGRAMS, FUNCTIONS AND SUBPROGRAMS

A FORTRAN program consists of a main program with or without subprograms. The main program and subprograms communicate with each other through parameters and common variables.

6.1 Main Program and Subprograms

A main program may be written with or without references to subprograms.

The PROGRAM statement may be used as the first statement of the main program.

PROGRAM name

name is an alphanumeric identifier from one to five characters; the first must be alphabetic. This statement is optional.

A main program may refer to both subroutines and functions which are compiled independently of the main program. A calling program is a main program or subprogram that refers to subroutines and functions.

6.2 Parameters

Main programs, subprograms, and functions use parameters as one means of communication. The parameters appearing in a subroutine call or a function reference are actual parameters. The corresponding parameters appearing with the subroutine or function name in the definition are formal parameters. Actual and formal parameters must agree in order, type and number.

6.2.1 Formal Parameters

The following are permissible forms for formal parameters:

array name

simple variable

function subprogram name

subroutine subprogram name

Since formal parameters are local to the subprogram containing them, they may be the same as names appearing outside the program unit.

No element of a formal parameter list may appear in a COMMON, EQUIVALENCE, or DATA statement within the subprogram. When a formal parameter represents an array, it should be declared in a DIMENSION statement within the subprogram. Otherwise, the loader writes an informative error message.

Example:

```
SUBROUTINE PER(A,I,X)
FUNCTION OLE(X)
```

A, I and X are formal parameters.

6.2.2 Actual Parameters

The following are permissible forms for actual parameters:

constant
simple or subscripted variable
arithmetic expression
array name
function subprogram name
subroutine subprogram name

When an actual parameter is a subroutine or function name, that name must also appear in an EXTERNAL statement in the calling program.

Example:

```
CALL PER(B,K,Y)
```

B, K and Y are actual parameters.

6.3 Function Subprogram

A function subprogram is a computational procedure which returns a single value associated with the function name. The mode of the function is determined by its name in the same way as a variable identifier.

The first statement of a function subprogram must have the following form:

FUNCTION $F(a_1, \ldots, a_n)$

F is the symbolic name of the function. The name of the function F must also appear as a variable name in the defining subprogram. The value of this variable at the time of execution of any RETURN statement in this subprogram is called the value of the function. The name of the function must not appear in any non-executable statement in the function subprogram except the FUNCTION statement. a_i are the formal parameters.

The function subprogram may contain any statement except SUBROUTINE, another FUNCTION statement, or any statement that directly or indirectly references the function being defined*.

Besides the FUNCTION F $(a_1, a_2, \ldots, a_n)$ statement where mode is determined by the first character, the following FUNCTION statements are accepted as alternate forms

INTEGER FUNCTION	F $(a_1, a_2, \ldots, a_n)$
REAL FUNCTION	F $(a_1, a_2, \ldots, a_n)$
DOUBLE PRECISION FUNCTION	F $(a_1, a_2, \ldots, a_n)$
COMPLEX FUNCTION	F $(a_1, a_2, \ldots, a_n)$
LOGICAL FUNCTION	F $(a_1, a_2, \ldots, a_n)$

F is the function name, and a_i are formal parameters. The type FUNCTION statement declares the type of the result returned by the function. Double integer functions may be declared by mentioning the function name in a type-statement list.

Example:

```
FUNCTION XSQ(A)
XSQ = A*A
RETURN
END
```

---

* In RT-FORTRAN, recursive calls are permitted.

6.3.1 Function Reference

$$F(a_1, \ldots, a_n)$$

F identifies the function being referenced. It is the same as the name in the FUNCTION statement. a_i are the actual parameters.

A function reference may appear any place in an expression where an operand may be used. The evaluated function will have a single value associated with the function name. When a function reference is encountered in an expression, control is transferred to the function indicated. When a RETURN or END statement in the function subprogram is encountered, control is returned to the statement containing the function, with the function reference replaced by the value of the function.

Example:

```
X = A+B*XSQ(D)
```

6.3.2 Function Parameters

The formal parameters of a function subprogram may not appear in either a COMMON, DATA or EQUIVALENCE statement in the function subprogram. When a function reference is executed, actual parameters are associated with all appearances of the corresponding formal parameters in executable statements and statement functions in the defining subprogram. If a formal parameter appears in a statement redefining its value, the corresponding actual parameter must be a simple or subscripted variable or an array name. A formal parameter may not appear in a redefining statement if a function reference associates it with another formal parameter in the same subprogram directly or via another element. If an actual parameter is an arithmetic expression, it is evaluated and its value is associated with the corresponding formal parameter.

If a formal parameter is an array name, the corresponding actual parameter must be an array. A formal parameter used as a format specification in a formatted READ or WRITE statement is assumed to be an array.

If an actual parameter is a function or subroutine name, the corresponding formal parameter must be used as a function or subroutine reference.

A function must have at least one, and not more than 32 parameters.

Examples:

1) Function Subprogram

```
      FUNCTION GREAT (A,B)
      IF(A-B)1,1,2
1     GREAT=A-B
      RETURN
2     GREAT=A+B
      END
```

Calling Program Reference

```
Z(I,J)=F1+F2-GREAT(C-D,3.*IJ)
```

2) Function Subprogram

```
      FUNCTION SYCHE(A,B,X)
      CALL X
      SYCHE=A/B*2.*(A-B)
      END
```

Calling Program Reference

```
      EXTERNAL EROS
      .
      .
      .
      R=S-SYCHE(TLIM,ULIM,EROS)
```

In the function subprogram, TLIM, ULIM replaces A,B. The CALL X is a call to a subroutine named EROS. EROS appears in an EXTERNAL statement so that the compiler recognizes it as a subroutine name rather than a variable identifier.

3) Function Subprogram

```
      FUNCTION A L(W,X,Y,Z)
      CALL W(X,Y,Z)
      AL=Z**4
      RETURN
      END
```

Calling Program Reference

```
EXTERNAL SUM
      .
      .
      .
G=AL(SUM,E,V,H)
```

In the function subprogram the name of the subroutine (SUM) and its parameters (E, V, H) replace W and X, Y, Z. SUM appears in the EXTERNAL statement so that the compiler will treat it as a subroutine name rather than a variable identifier.

6.4 Statement Functions

Statement function definitions must precede the first executable statement of the program or subprogram and must follow any specification statements. The name of a statement function must not appear in an EXTERNAL statement, nor as a variable name or an array name in the same program or subprogram. A statement function applies only to the program or subprogram containing the definition; it is defined by a statement of the form:

$$f(a_1, a_2, \ldots, a_n) = e$$

f is the statement function name, e is any expression. a_i are variable names which are dummy arguments indicating type, number, and order of arguments; they may be the same as variable names of the same type appearing elsewhere in the program unit. n may not exceed 32. f and e must be both logical or both non-logical.

Examples:

1.
```
LOGICAL C,P, EQV
EQV(C,P) = (C.AND.P).OR.(.NOT.C.AND..NOT.P)
```

2.
```
COMPLEX D,F
D(A,B) = (3.2,0.9)*EXP(A)*SIN(B)+(2.0,1.)*EXP(Y)*COS(B)
```

3.
```
GROS(R,HRS,OTHER) = R*HRS + R*.5*OTHER
```

6.5 Library Functions

Function subprograms that are used frequently have been written and stored in a reference library and are available to the programmer through the compiler.

A list of these functions is found in Appendix D. When a reference appears in the source program, the compiler identifies it as a library function and generates a calling sequence within the object program.

Example:

X = SIN(A)+A LOG(B)

6.6 EXTERNAL Statement

When the actual parameter list of a given function or subroutine reference contains a function or subroutine name, that name must be declared in an EXTERNAL statement. Its form is

EXTERNAL $name_1$, $name_2$, $name_3$, ...

$name_i$ is a function or subroutine name used as a parameter.

The EXTERNAL statement must precede the first executable statement in any program in which it appears. When it is used, EXTERNAL always appears in the calling program. (See examples in Section 6.3.2.)

6.7 Subroutine Subprograms

A subroutine is a computational procedure which may return none, one, or more values. No value or type is associated with the name of a subroutine. The first statement of a subroutine subprogram must be one of the following:

SUBROUTINE s

SUBROUTINE s $(a_1, \ldots, a_n)$

S is an alphanumeric identifier; a_i are formal parameters and may be variable names, array names, or subprogram names.

The name of the subroutine must not appear in any other statement in the subprogram. The names of the formal parameters a_i may not appear in a COMMON or DATA statement in the subprogram. The parameters may be defined or redefined within the subprogram so that they may effectively return results.

No value is associated with the name of the subroutine, and the subroutine must be referenced by a CALL statement.

Rules:

1) The name of the subroutine may not appear in any declarative statement (TYPE, DIMENSION) in the subroutine.

2) The name of the subroutine must never appear within the subroutine as an identifier in a replacement statement, in an input/output list, or as an argument of another CALL*.

3) No element of a formal parameter list may appear in a COMMON, EQUIVALENCE, DATA, or EXTERNAL statement within the subroutine.

4) When a formal parameter represents an array, it should be declared in a DIMENSION statement within the subroutine. If it is not declared, a loader error will result.

5) The SUBROUTINE statement may have from zero to 32 formal parameters.

6.8 CALL Statement

The executable statement in the calling program to refer to a subroutine is one of the forms:

```
CALL S
CALL S (a1,...,an)
```

S is the name of the subroutine being called, and a_i are actual parameters. The name may not appear in any specification statement in the calling program except in EXTERNAL statement. A subroutine may also be referenced by the appearance of its name in an EXTERNAL statement.

The CALL statement transfers control to the subroutine. When a RETURN or END statement is encountered in the subroutine, control is returned to the next executable statement following the CALL in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the loop is satisfied.

Examples:

1) Subroutine Subprogram

```
SUBROUTINE TEST (X,Y,Z)
Z=2*X+X/Y
END
```

---

* Note that in RT-FORTRAN, the subroutine name may appear in a CALL statement both as parameter and as subroutine name.

Calling Program References

```
CALL TEST(X(I),Y(I),A)
      .
      .
      .
CALL TEST(A,B,C)
      .
      .
      .
CALL TEST(X(I)+H, Y(1)+2., W)
```

2) Subroutine Subprogram (Matrix Multiply)

```
SUBROUTINE MATM
COMMON/BLK1/X(20,20),Y(20,20),Z(20,20)
DO 10 I=1,20
DO 10 J=1,20
Z(I,J)=0
DO 10 K=1,20
10 Z(I,J)=Z(I,J)+X(I,K)*Y(K,J)
RETURN
END
```

Calling Program References

```
COMMON/BLK1/A(20,20),B(20,20),C(20,20)
      .
      .
      .
CALL MATM
      .
      .
      .
```

3) Subroutine Subprogram

```
SUBROUTINE HTAR(Y,Z)
COMMON/1/X(100)
Z=0
DO 5 I=1,100
5 Z=Z+X(I)
CALL Y
RETURN
END
```

Calling Program Reference

```
COMMON/1/A(100)
EXTERNAL PRNT
      .
      .
      .
CALL HTAR (PRNT,SUM)
```

4) Subroutine Subprogram

```
SUBROUTINE PIP (A,B,C)
A=B**C
   .
   .
   .
END
```

Calling Program Reference

```
      .
      .
      .
CALL PIP (V(1),X,3)        parameter must agree
                           in number
```

6.9 Program Arrangement

NORD STANDARD FORTRAN assumes that all statements and comments appearing between a PROGRAM, SUBROUTINE, or FUNCTION statement, or the first statement of a main program and an END statement belong to one program unit. A program unit must consist of at least one executable statement that is actually executed. Any specification statements or statement function definitions must precede the first executable statement with specifications preceding statement function definitions. FORMAT statement may appear anywhere in a program unit. The last executable statement in a main program or subprogram must be one of the following:

```
STOP
RETURN
END
```

A subprogram normally contains RETURN statements that indicate the end of logic flow within the subprogram and return control to the calling program. In a function subprogram, control returns to the statement containing the function reference at which time the value of the function is made available to the calling program. In subroutine subprograms, control returns to the next executable statement following the CALL statement. A STOP statement in the main program causes an exit to the operating system.

END is the final statement in a program or a subprogram. In a subprogram, END causes a return to the calling program and may replace a final RETURN statement.

A typical arrangement of a set of main program and subprograms follows.

```
PROGRAM TEST
      .
      .
      .
END
SUBROUTINE S1
      .
      .
      .
END
SUBROUTINE S2
      .
      .
      .
END
      .
      .
      .
FUNCTION F1 (...)
      .
      .
      .
END
FUNCTION F2 (...)
      .
      .
      .
END
```

6.10 RETURN and END Statements

A subprogram normally contains one or more RETURN statements that indicate the end of logic flow within the subprogram and return control to the calling program. The form is

```
RETURN
```

In function references, control returns to the statement containing the function. In subroutine subprograms, control returns to the calling program.

The END statement marks the physical end of a program, subroutine subprogram or function subprogram. If the RETURN statement is omitted, END acts as a return to the calling program.

A main program must not contain a RETURN statement.

6.11 RT-Program Statement

By using the RT-program statement, the user can generate an RT-description for his program. This program may be executed in the same way as all other RT-programs written in assembly code (see the SINTRAN III Users' Guide for further information). The RT-statement has the following format:

PROGRAM <prog. name> , <priority>

The <prog. name> may be any acceptable FORTRAN name. It will be referred to in the loader tables and must be defined only once. The <priority> specifies the priority of the RT-program and may be any unsigned number between 1 and 225. An example might be:

PROGRAM PER, 5

Here PER will be defined to a real-time program with a priority of 5.

The <priority> may be omitted. Then the <priority> will be set to zero, and a warning message will be printed at load-time.

7 I/O STATEMENTS

Input/output statements control the transfer of information between the computer memory and logical units, which can be external devices or mass storage files.

7.1 READ/WRITE Statements (Formatted)

The following definitions for i, n, and L apply for all I/O control statements.

The logical unit number, i, must be an integer variable, an integer constant. or an array name. (ENCODE-/DECODE-effect.)

In case the device number is an array name, the execution of a WRITE statement will cause the list elements to be placed in the array according to the FORMAT statement.

The FORMAT statement describing the format of the data is represented by n which must be a statement label number or an array name.

The input/output list is specified by L.

ERR=l_1 is an optional clause that is used to transfer control to statement label l_1 if an error is detected in the execution of the input/output statement. The statement label l_1 must be contained in the same routine as the input/output statement.

END=l_2 is an optional clause that is used to transfer control to statement label l_2 if, during the execution of an input/output statement, and end of file is encountered on input or the end of a mass storage file is encountered on output. The statement labeled l_2 must be contained in the same routine as the input/output statement.

7.1.1 WRITE Statement

WRITE (i,n,ERR=l_1) L

This statement transfers information from storage locations given by identifiers in the list (L) to a specified unit (i) according to the FORMAT statement (n).

A logical record containing up to 136 ASCII characters are output to the unit. The number of words in the list (L) and the FORMAT statement (n) determines the number of records that will be written on a unit. If the logical record is less than 136 characters, the record will be terminated with the last data item in the record.

Examples:

```
      DIMENSION D(10,10)
      WRITE (5,10) A,B,C
10    FORMAT (3F10.5)
        .
        .
        .
      WRITE (5,10)((D(I,J),I=1,N1),J=1,N2)
      WRITE (5,10) D
      WRITE (5,20)
20    FORMAT (6X,5HABCDE)
```

7.1.2 READ Statement

READ (i,n,ERR=l_1,END=l_2) L

This statement transfers information from a specified unit (i) into storage locations named by the list (L) identifiers according to FORMAT statement (n).

The number of words in the list and the format specifications must conform to the record structure on the logical unit, (up to 136 characters per record).

Examples:

```
      READ (4,10,ERR=98,END=99) X,Y,Z
10    FORMAT (3F10.5)
98    STOP 1
99    STOP 2


      DIMENSION D(10,10)
      READ (4,11) ((D(I,J),I=1,N1),J=1,N2)
11    FORMAT (5E12.2)
      READ (4,11) D
      READ (4,20)
20    FORMAT (6X,10HINPUT␣DATA)
```

7.2 INPUT/OUTPUT Statements

The execution of these statements cause the transmission of data to conform to a standard FORMAT E16.8 for real list items, and I16 for integer list items.. (See Section 8.4.)

7.2.1 OUTPUT Statement

OUTPUT (i,ERR=l_1) L

This statement transfers information from storage locations given by identifiers in the list (L) to a specified unit (i) according to a standard FORMAT, (see Section 8.4.2). Else the rules are as for WRITE statement.

Example:

```
OUTPUT (5) A,B,I1,X

DIMENSION D(10)
OUTPUT (5) D
```

7.2.2 INPUT Statement

INPUT (i,ERR=l_1,END=l_2) L

This statement transfers information from a specified unit (i) into storage locations named by the list (L) identifiers according to a standard FORMAT (see Section 8.4.1). Else the rules are as for a READ statement.

Example:

```
INPUT (4) A,B,C

DIMENSION D(5)
INPUT (4) D

    or

INPUT (4) (D(I),I=1,5)
```

7.3 Binary Input/Output

The binary transmission mode transports bit patterns from one place to another, e.g., from an external device into central memory or reverse.

One half-word (8 bits) is moved at a time. Two neighbour half-words are placed side by side in one memory location. The order of the half-words is preserved during the transfer.

7.3.1 Write Binary

WRITE (i,ERR=l_1) L

This statement transfers information from storage locations given by identifiers in the list (L) to a specified external unit (i) in binary mode.

Example:

```
DIMENSION IA(50),B(10)
WRITE (3) IA,B
```

7.3.2 Read Binary

READ (i,ERR=l_1,END=l_2) L

This statement transfers information from the specified unit (i) into storage locations named by the list (L) identifiers in binary mode.

Example:

```
DIMENSION IA(50),B(10)
READ (2) IA,B
```

7.4 Transmission of Arrays

Part or all of an array can be represented as a list item. Multi-dimensional arrays may appear in the list, with values specified for the range of the subscripts in an implied DO loop.

7.4.1 Implied DO Loop

The general form is:

$(((A(I,J,K),B(I,J,K),\gamma_1=m_1,m_2,m_3),\gamma_2=n_1,n_2,n_3),\gamma_3=p_1,p_2,p_3)$

where

A,B	are array names,
m_i, n_i, p_i	are unsigned constants or predefined positive integer variables. If m_3, n_3 or p_3 is omitted, it is construed as 1.
I,J,K	are subscripts of A and B and must be integer variables or constants.
$\gamma_1, \gamma_2, \gamma_3$	are I, J, or K; $\gamma_1 \neq \gamma_2 \neq \gamma_3$

The I/O list (L) may contain five nested implied DO loops.

Example:

As an element in an input/output list, the expression

WRITE(i)(((A(I,J,K), I=m_1,m_2,m_3), J=n_1,n_2,n_3), K=p_1,p_2,p_3)

implies a nest of DO loops of the form

```
DO 10      K = p1,p2,p3
DO 10      J = n1,n2,n3
DO 10      I = m1,m2,m3
           WRITE(i) A(I,J,K)
10 CONTINUE
```

(Be aware that the last way of writing will generate more output records, as the WRITE generates at least one record every time it is executed!)

Example:

To write the elements of a 3 by 3 matrix by columns:

((A(I,J), I=1,3), J=1,3)

To write the elements of a 3 by 3 matrix by rows:

((A(I,J), J=1,3), I=1,3)

Example:

For example, a multi-dimensional non-subscripted list element, SPECS, with an associated DIMENSION SPECS (8,6,4) statement is transmitted as if under control of an implied DO loop:

WRITE(i,n) SPECS

is equivalent to:

WRITE(i,n)(((SPECS(I,J,K), I=1,8), J=1,6), K=1,4)

7.5 Addressing Records on Files (SINTRAN III)

On files opened for sequential read-write (RW) the initial record number of the I/O transfer may be specified in the READ/WRITE statements.

Example:

```
READ(i/k ,n)L
```

This statement transfers information from the kth record of the file i into storage locations named by the list (L) identifiers according to FORMAT statement n.

7.6 Mass Storage Statements

To simplify file handling, mass storage statements are provided. The logical unit number, i, is an integer variable or an integer constant.

7.6.1 REWIND Statement

```
REWIND i
```

Moves the file pointer to the beginning of logical unit no. i. When the file pointer is already at the beginning of logical unit no. i, the statement acts as a do-nothing statement.

7.6.2 BACKSPACE Statement

```
BACKSPACE i
```

Backspaces the pointer one logical record in unit no. i. When the pointer is already at the beginning of file no. i, the statement acts as a do-nothing statement.

7.6.3 ENDFILE Statement

```
ENDFILE i
```

Writes an end-of-file mark on logical unit i.

7.7 Additional Mass Storage Utility Subprograms (SINTRAN III)

Note: If the following subprograms are declared as integers and treated like functions they will return the value zero if no errors occurred during the transfer, else the error-code. Also the appropriate error-message will appear on the terminal.

Example:

```
INTEGER SETBL

IERR = SETBL (65,512)
IF (IERR.NE.0)    GOTO 100
```

7.7.1 Open a File

INTEGER OPEN

IERR=OPEN('< filename>< blank>', < connected file no> , <access code>
or
CALL OPEN('< filename>< blank>', < connected file no.> , <access code>

File name:	as in NORD File System
Connected file no:	logical file number applied by the user in his READ/WRITE statements (must be different from 1).
Access code:	0 - sequential write 1 - sequential read 2 - random read or write 3 - random read 4 - sequential read or write 5 - sequential write append 6 - random read or write common 7 - random read common

The specified file is opened for access. The call acts like the OPEN-FILE command (see NORD File System manual).

7.7.2 Close a File

INTEGER CLOSE

IERR = CLOSE (<file no.>)
or
CALL CLOSE (<file no.>)

This call will close the file with the specified file number. If the number is -1, all files for entered user are closed.

7.7.3 Read (Random) Part of a File

INTEGER RFILE

IERR = RFILE (<file no> , < return flag > , < core address > ,
< block no.> , < no. of words >)
or
CALL RFILE (<file no> , < return flag > , < core address >,
<block no. > , < no. of words >)

This is a subroutine to read a random record from a file. file no identifies the file. If return flag is zero, the program will be set in a wait state until the transfer is finished. If return flag is set non-zero, there will be return from RFILE as soon as the transfer is started, so that the program and the transfer can proceed in parallel.

The parameter core address determines where the record should be placed. In FORTRAN this can be any array name. block number gives the file block number where the record starts, while number of words defines the record size. There is no inherent restriction on the record size.

7.7.4 Write (Random) Part of a File

INTEGER WFILE

IERR = WFILE (< file no. > , < return flag >, <core address>
< block no > , < no. of words >)
or
CALL WFILE (<file no. > , < return flag > , < core address > ,
< block no > , < no. of words >)

This is a subroutine to write a random record onto a file. The parameters have the same meaning as for RFILE. If the record does not fill the last block completely, the rest of the block will have undefined contents.

7.7.5 Set Block Size of a File

INTEGER SETBS

IERR = SETBS (< file no. > , < block size >)
or
SETBS (< file no.> , < block size >)

This call will set the block size of the specified opened file. The block size may be any number greater than or equal to 1 (default = 256 words).

7.7.6 Set Byte Pointer of a File

INTEGER SETBL

IERR = SETBL (< file no. > , < byte number >)
or
CALL SETBL (< file no. > , < byte number >)

This call will set the byte pointer of the file to the specified byte number. The call may be applied on files opened for RW only.

7.7.7 Set Block Pointer of a File

INTEGER SETBT

IERR = SETBT (< file no. > , < block number >)
or
CALL SETBT (< file no. > , < block number >)

This call will set the byte pointer of the file to the first byte in the specified block.

7.7.8 Read Byte Pointer of a File

INTEGER REABT

IERR = REABT (< file no. > , < byte number read >)
or
CALL REABT (< file no. > , < byte number read >)

This call will return in second parameter the current byte pointer of the opened file.

8 FORMAT SPECIFICATIONS

8.1 Introduction

The FORTRAN FORMATTED INPUT/OUTPUT System, FIO, is completely re-entrant and can therefore be used (shared) by several different programs on different priority levels simultaneously.

The FIO-System has three different "modes" of transmission of data between an external device and computer memory.

8.1.1 Formatted Input/Output

This is the general FORTRAN input/output whereby the data transmission is performed under control of a FORMAT statement.

Example:

```
WRITE (5,10) A,B,C,K,L,M
10 FORMAT (2E20.8, I15, F5.1,/,2X,2I10)
```

where (5,10) specifies the logical unit no. 5 (see Appendix H) and FORMAT statement no. 10 and A,B,....,M is the I/O-list.

Note that the list item and the format specification should normally be of the same type, but they can also be of different types. A list item of integer type can be input or output under F or E specification, and a list item of real type can be input or output under I specification. (On input the data string is processed according to the format specification before it is converted to the type of the list item. This feature should therefore be used with caution.)

See Appendix H for I/O device numbers.

8.1.2 Binary Input/Output

This is also a standard FORTRAN feature. Transmission in this mode will merely move the data from one place to another (specified by the programmer) without conversion.

Example:

```
READ (2) L
```

where (2) specifies logical unit no. 2 and L is the I/O-list.

8.1.3 "Free" Format Input/Standard Format Output

Transmission of data in this mode includes conversion of data similar to that of formatted I/O. But in using this form of I/O, the programmer need have no concern about the FORMAT statement since the data conversion is not under external format control.

Example:

```
INPUT     (2) A,B,C,D,K,L,R
OUTPUT    (3) A,B,C,D,K,L,R
```

8.2 Formatted Input/Output

FORTRAN READ and WRITE statements of the form

```
READ  (i,n) L
WRITE (i,n) L
```

cause the generation of calls to the formatted I/O routine. The form of these calls is shown in the subroutine specification, "Formatted Input/Output". In the above statements i is a logical unit number, n is a FORMAT statement number, and L is the I/O list.

8.2.1 FORMAT Statement

The FORMAT statement is used to specify the conversion to be performed on data being transmitted during formatted (BCD) input/output. It is non-executable and may be placed anywhere in the program. In general, conversion performed during output is the reverse of that performed during input. FORMAT statements have the form

$$\text{FORMAT } (s_1, s_2, s_3, \ldots\ldots\ldots, s_n)$$

where

$n \geq 0$, and
s_1 has either a formatted specification of one of the forms described below or a repeated group of such specifications in the form

$$r(s_1, s_2, \ldots\ldots, s_m)$$

where $m > 0$, r is a repeat count (described below), and s_1 has one of the format specifications listed below.

Format specifications describe the kind or type of conversion to be performed, specific data to be generated, and editing to be executed. Each integer or real entity appearing in an input/output is processed by a single format specification.

8.2.2 Record

A record is a unit, composed of a number of positions or other smaller units. A NORD-record has variable length, i.e. from one LF to the next CR. The maximum record length has 136 positions. FORMAT statements define records. The first left parenthesis starts a new record, while the last right parenthesis terminates it. The number of positions in each record must not exceed the maximum number, but may be less than it.

Note: The right parenthesis of a parenthesized specification group, not preceded by a repetition factor, causes termination of a record.

Example: The program:

```
      PROGRAM T1
      DIMENSION A(5)
      DO 1  I=1,5
    1 A(I)=10.0*I
      DO 2  J=1,5
    2 WRITE(1,3) J,(A(I),I=1,5)
    3 FORMAT(2X,I2,(4X,F5.1))
      END
```

causes the following output:

```
 1      10.0
       20.0
       30.0
       40.0
       50.0
 2      10.0
       20.0
       30.0
       40.0
       50.0
 3      10.0
       20.0
       30.0
       40.0
       50.0
 4      10.0
       20.0
       30.0
       40.0
       50.0
 5      10.0
       20.0
       30.0
       40.0
       50.0
```

8.2.3 FIO-Conversion Specifications

rFw.d	Real number without exponent
rEw.d	Real number with exponent
rDw.d	Double Precision number with exponent
rIw	Integer or double integer
rAw	Alphanumeric specification
rZw	Octal integer specification
rLw	Logical specification
Tw	Tab-specification
±nP	Scaling factor

Editing specifications:

rX	Intra-line spacing
nHs	Text
....	Text
'....'	Text
r/	New record

The letters r, w, d, n, and s in the specifications above have the following meanings:

r	is an optional, unsigned integer that indicates that the specification is to be repeated r times. When r is omitted, its value is assumed to be 1. For example, 3I6 is equivalent to I6, I6, I6. For X specification, r must be defined.
w	is an unsigned integer that defines the width, in characters (including digits, decimal points, algebraic signs and blanks), of the external representation of the data being processed.
d	for F, E and D specifications, is an unsigned integer that specifies the number of fractional digits appearing in the magnitude portion of the external field.
n	is an unsigned integer that defines the number of characters being processed.
s	is a string of characters acceptable to the FORTRAN processor.

8.2.3.1 F Format (Fixed Decimal Point)

Form: rFw.d

Real data may be processed by this form of conversion. The total width of the field, including decimal point and sign, if any, is specified by w, and the value of d allows for the appropriate number of digits in the fractional portion of the field. F format specification should be used for numbers that range from 1.0E-10 to 1.0E10 in absolute value.

OUTPUT

Internal value are rounded to d decimal places with an over all length of w. The field is right-justified with as many leading blanks as necessary. Negative values are preceded with a minus sign. Consequently, for the specification F11.4,

273.4	is converted to	273.4000
7	is converted to	7.0000
-.003	is converted to	-.0030
-442.30416	is converted to	-442.3042

If a value requires more positions than are allowed by the magnitude of w, the output field is filled with asterisks. This happens if

$$w < d+2+n$$

where

n is the number of digits to the left of the decimal point.

INPUT

Input strings may take any of the integer or real constants forms discussed below in Section 8.2.4, "Numeric Input Strings". Each string will be of length w with d characters in the fractional portion of the value. If a decimal point is present in the input string, the value of d is ignored, and the number of digits in the fractional portion of the value will be explicitly defined by that decimal point. For the specification F10.3,

33	is converted to	.033
802142	is converted to	802.142
.34562	is converted to	.34562
-7.001	is converted to	-7.001

8.2.3.2 E Format (Normalized with Exponent)

Form: rEw.d

Real data is processed by this form of conversion.

OUTPUT

Internal values are converted to real constants of the forms

$$d.ddd.....dE^{\pm}ee$$

where the length of the output field is w, and the number is scaled to have one digit of the mantissa to the left of the decimal point, such that the number of digits in the mantissa is d+1. The exponent, $^{\pm}$ee, is interpreted as a multiplier of the form $10^{\pm ee}$.

Internal values are rounded to d+1 digits, and negative values are preceded by a minus sign. The external field is right-justified and preceded by the appropriate number of blanks. The following are examples for the specification E15.7

```
    90.4450 is converted to  9.0445000E+01
-435739015 is converted to -4.3573902E+08
   .000375 is converted to  3.7500000E-04
        .2 is converted to  2.0000000E-01
       0.0 is converted to  0.0000000E+00
```

The field is counted from the right and includes the two exponent digits, the sign, the letter E, the fractional digits, the decimal point, the most significant digit, and the sign of the value (minus or space). If a width specification is of insufficient magnitude to allow expression of an entire value, w < d+7, the field will be filled with asterisks. E format can be used for numbers that range from 1.0E-100 to 1.0E100 in absolute value.

INPUT

The discussion in Section 8.2.4 contains a description of the form permissible for strings of input characters. Conversion is identical to F format conversion. In particular, input fields for conversion in E format need not have exponents specified.

Examples:

Input Value	Specification	Converted to
-113409E2	E11.6	-11.340900
-409385E-03	E11.2	-4.09385
849935E-02	E10.5	.0849935
6851	E4.0	6851.0

First the decimal point is positioned according to the specification; then, the value of the exponent is applied to determine the actual position of the decimal point. In the first example, -113409E2 with a specification of E11.6 is interpreted as -.113409E02, which when evaluated (i.e., $-.113409 * 10^2$), becomes -11.340900.

8.2.3.3 D Format (Normalized with Exponent)

This format is equivalent to the E format. It is also used in the same way.

8.2.3.4 I Format (Integer or double integer)

Form: rIw

Integer data is processed by this form of conversion.

OUTPUT

Internal values are converted to integer constants, w giving the maximum number of digits to be output. Negative values are preceded by a minus sign, and the field will be right justified and preceded by the appropriate number of blanks. The specification I6 implies that:

273	is converted to	273
7	is converted to	7
-24204	is converted to	-24204

If the magnitude of data requires more positions than are permitted by the value of the width w, the field will be filled with asterisks. I format can be used for integer numbers that range from -32768 to 32767. I format can also be used for real numbers.

INPUT

External input strings must take the integer form discussed in Section 8.2.4.

8.2.3.5 A Format (Alphanumeric)

Form: rAw

OUTPUT

Internal binary values are converted to character strings at the rate of eight binary digits (two hexadecimal digits) per character. The more significant characters are converted first. That is, conversion is from left to right, at the rate of two characters per word. Note that when the magnitude of w does not provide for enough positions to express the data value completely, the external field is shortened from the right (least significant) portion. This is not treated as an error condition. When w has a value greater than necessary, the external character string is preceded by the appropriate number of blank characters.

For example,

Internal Value	Specification	Output
HI	A2	HI
HO	A3	␣HO
!X	A1	!

INPUT

Let v=2 (integer) or v=6 (real).
When the width w is larger than necessary (that is, w > v), the list item is filled with the rightmost characters. For example, if the list item is integer type, and the specification A10 is used, ABCDEFGHIJ is converted to IJ alone. However, when the value of w is less than v, the more significant positions of the list item are filled with w characters, and the remainder of the positions are filled with blanks. Q, with a specification of A1, is converted to Q␣ if the list item is an integer.

8.2.3.6 H Format (Hollerith)

Form: nHs

OUTPUT

The n characters in the strings are transmitted to the external record. For instance,

Specification	External string
1HE	E
7H␣␣VALUE	␣␣VALUE
7HKR␣3.95	KR␣3.95
9HX(2,5)␣=␣	X(2,5)␣=␣

INPUT

n characters from the input record are inserted in the format string following the nH specification.
For example:

Specification	Input string	Resultant Spec.
3H123	ABC	3HABC
5HTRUE␣	FALSE	5HFALSE
6H␣␣␣␣␣␣	RANDOM	6HRANDOM

This feature can be used to change titles, dates, column headings, and so forth, that are to appear on a record generated by the H specification.

8.2.3.7 *...Text...* or '...Text...'

This specification may be used instead of nH to input or output text from a format. The * 's mark the ends of the Hollerith field. Note that an * should not be included in an input string under * specification. Comma is optional after an * .. * specification.

Example:

FORMAT (*HOLLERITH*)

8.2.3.8 X Format (Skip)

The form of the X specification is

rX

where r must be ≥ 1.

OUTPUT

The next r positions in the output record will be blanks. In other words, a field of r blanks will be created. For example, the specifications

4HWXYZ, 4X, 4HIJKL

generate the following external string:

WXYZ␣␣␣␣IJKL

INPUT

The next r characters from the input string are ignored (that is, they are skipped). For example, with the specifications

F5.2, 6X, I3

and the input string

76.41IGNORE697

the characters

IGNORE

will not be processed.

8.2.3.9 T Format (Tab)

Form: Tw

This specification causes processing to continue at the w'th character of the input or output record.

8.2.3.10 Z Format (Octal)

Form: rZw or rOw

Octal input/output can be performed specifying any of the data types - integer or real - in the I/O list.

As each octal digit represents three bits, and the NORD-1 wordlength is sixteen bits, the following connection is used:

Integers : treated as one 16 bit word, 6 octal digits

Reals : treated as one 48 bit word, 16 octal digits

OUTPUT

Internal binary values are converted to character strings at a rate of three bits per character.

Integers : If w ≥ 6, the leftmost digit is the value of the leftmost bit of the word.

Reals : If w ≥ 16, the three words are treated as a single forty-eight bit word.

Note that when the magnitude of w does not provide for enough positions to express the data value completely, the most significant digits are truncated. This is not treated as an error condition. When w has a value greater than necessary, the external character string is preceded by the appropriate number of blank characters.

Example:

Specification	Internal value			Output value
	Integers:			
Z8	137420			137420
Z5	137420			37420
Z3	040001			001
	Reals:			
Z16	040003	100000	000000	2000130000000000
Z11	040003	100000	000000	30000000000

INPUT

w characters from the input record are assembled into the list item at a rate of three bits per character.

If w < 6 for integers, and w < 16 for reals, the input characters will be right justified, and the leftmost part will be filled with zeros.

If w > 6 for integers, and w >16 for reals, the list item will be filled with the rightmost characters.

Example:

Specification	Input value	Internal value		
		Integers:		
Z6	137326	137326		
Z6	␣␣2671	002671		
Z8	37533235	133235		
Z2	35	000035		
		Reals:		
Z16	2000130000000002	040003	100000	000002

8.2.3.11 L Format (Logical)

Form: rLw

This code is used only with input and output of logical variables.

If Lw is specified for output and the value of the logical list item is .TRUE., the rightmost position of the field with length w contains the letter T. If the value is .FALSE., the letter F is printed, instead.

On input, the field width is scanned from left to right for the first occurrence of T or F, and the value of the corresponding logical list item is set to .TRUE. or .FALSE., respectively. All other characters of the external input field are ignored. In the absence of T or F in the input field, no value will be stored.

8.2.3.12 / Specifications (Record Separator)

The form of the / specifications is

r/ or /

Each slash (/) specified causes another record to be processed. In the case of continuous specifications (i.e., ////.../ or r/), records are ignored during input (since no conversion occurs between each of the slash specifications), and blank records are generated during output operations. The same condition can occur when a slash specification and either of the parenthesis characters surrounding the field specifications are continuous, (i.e., r(/)). A slash preceding the final right parenthesis in a FORMAT statement is not ignored.

OUTPUT

Whenever a slash specification is encountered, the current record being processed is output, and another record is begun. If no conversion has been performed when the slash is encountered, a blank record is created. The statements

```
      WRITE (5,10) X,K
10 FORMAT (F5.3//I13)
```

are processed in the following manner:

1) A record is begun, and X is converted with the specification F5.3.

2) The first slash is encountered, the record containing the external representation of X is terminated, and another record is begun.

3) The second slash is encountered, the second record is terminated, and a third record is started. Note that since no conversion occurred between the termination of the first and second records, the second record was blank.

4) The value of the variable K is converted with the I13 specification, the closing right parenthesis is encountered, and the third record is terminated.

If a third item, Z, were added to the output list, as in

```
WRITE (5,10) X,K,Z
```

the following additional steps will occur:

5) A fourth record is begun, and Z is converted using the specification F5.3.

6) The first slash is re-encountered, the fourth record is terminated, and a fifth record is begun.

7) Again, the second slash is processed; the fifth record, which is blank, is terminated, and the sixth record is started.

8) Since there are no more list items, the specification I13 is not processed, a termination occurs, and the final or sixth record, which is also blank, is output.

Note that the processing of Z in steps 5) through 8) is equivalent to processing with the statement:

```
10 FORMAT (F5.3,//)
```

since the specification I13 was not utilized.

The original FORMAT statement could also have been written as

```
10 FORMAT (F5.3,2/I13)
```

or

```
10 FORMAT (F5.3,2/,I13)
```

both of which would cause identical effects.

The two statements

```
      WRITE (5,4) X
4 FORMAT (3/E12.4/)
```

cause the generation of the three blank records, followed by a record containing the value of X (converted by the specification E12.4), followed by another blank record.

INPUT

The effect of slash specifications during input operations is similar to the effect for output, except that for input, records are ignored in the cases where blank records are created during output. For example, the statements:

```
      READ (5,4) X
4 FORMAT (3/E12.4/)
```

cause three records to be bypassed, a value from the fourth record to be converted (with the specification E12.4) and assigned to X, and a fifth record to be bypassed. This means that, as with the last example for output, records created with a FORMAT statement containing slash specifications can be input by use of the identical FORMAT statement. This is not true in FORTRAN systems that ignore a final slash.

8.2.3.13 Scale Factor

Form: $\pm$nP

This specification effects only E and F output and has no effect on input.

Output: $\pm$nP in front of

Iw : no effect

Fw.d : (external value) = (internal value) $\cdot 10^{\pm n}$

n is an arbitrary integer, $n \leq 99$. The + sign in front of n is optional.

Example: internal value = 3.1456789

Specification	Output	Comment
F10.3	␣␣␣␣3.146	
1PF10.3	␣␣␣31.457	
4PF10.3	␣31456.789	
6PF10.3	**********	Too short field
-1PF10.3	␣␣␣␣␣.315	
-3PF10.3	␣␣␣␣␣.003	
-4PF10.3	**********	Too short field

Ew.d : (external value) = (internal value)

The mantissa of the output is multiplied by $10^{\pm n}$ and $\pm$n is subtracted from the exponent part. The $\pm$nP specification is valid for the specification (E or F) it is placed in front of: For instance, in the format

(5P6F15.3, F10.2)

the 5P scaling factor will have effect on the six real numbers output by the 6F15.3 specification only, and the last number output by F10.2 will <u>not</u> be scaled.

Example: internal value = -3.1456789

Specification	Output	Comment
E15.3	␣␣␣␣-3.146E+00	
4PE15.3	␣␣31456.789E-04	
6PE15.3	**********	Too short field
-3PE15.3	␣␣␣␣-0.003E+03	
-4PE15.3	**********	Too short field

8.2.3.14 Parenthesized Format Specification

Within a FORMAT statement any number of specifications may be repeated by enclosing them in parentheses, preceded by an optional repeat count, in the form shown below.

$$r(s_1, s_2, s_3, \ldots\ldots\ldots, s_m)$$

where m > 0. For example, in processing the statement

```
3 FORMAT (3(A4, F5.2, 3X), 3I10)
```

each repetitive specification is exhausted in turn, as in each singular specification. The following are additional examples of repetitive specifications:

```
34 FORMAT (4X, 2(A8, 1X, 7E12.3), I4, 3(I2, I5))
1125 FORMAT (/A4, F10.7, 5(E14.4, 2/) E14.5)
```

Nesting of this type is permissible to a depth of two levels. The presence of parenthesized groups within a FORMAT statement affects the manner in which the FORMAT is rescanned if more list items are specified than are processed the first time through the FORMAT statement. In particular, when one or more such groups have appeared, the rescan begins with the group whose right parenthesis was the last one encountered prior to the final right parenthesis of the FORMAT statement.

8.2.4 Numeric Input Strings

A numeric input string consists of a string of digits with or without a leading sign, decimal point, or trailing exponent. An exponent is normally specified as

$$E^{\pm}e$$

where the plus sign is optional and e is a one- or two-digit number. The form $^{\pm}e$ is also accepted (without the E), in which case the plus sign is not optional. Thus, a variety of forms may be used to express data for numeric input, such as

±n	±n.m	±n.	±.m
±nE±e	±n.mE±e	±n.Ee	±.mE±e
±n±e	±n.m±e	±n.±e	±.m±e

where the plus signs are optional except in an exponent field without an E (as described above).

Note: The form ±n is the only form accepted by an I specification. All are accepted by E and F specifications.

The field terminates only when the width is exhausted or by a comma or CR. The following rules apply to blanks in numeric fields with a width specified:

1) Leading blanks are ignored, except that they are counted as part of the field width.

2) Once any non-blank character has been found, all blanks beyond that point are treated as zeros.

For a format specification such as F10.0, all the input strings in each of the columns below produce the value shown in the top line of the column. The first three lines in each column are typical numeric fields; the others are permissible but less readable.

-.004	7.5E12	0
-4E-3	.75E+13	0.0
-.004	75E11	
-␣␣␣␣␣4␣-4	75␣␣␣␣␣E6	0 + 0
.␣␣4E	750+10	0E
-4␣␣␣␣␣-8	.␣␣␣75E16	+ -

On input, a plus sign for the exponent field following an E is optional.

8.2.5 FORMAT and List Interfacing

Formatted input/output operations are controlled by the FORMAT requested by each READ or WRITE statement. Each time a formatted READ or WRITE statement is executed, control is passed to the FORMAT processor. The FORMAT processor operates in the following manner:

1) When control is initially received, a new input record is read, or construction of a new output record is begun.

2) Subsequent records are started only after a slash specification has been processed (and the preceding record has been terminated), or after the final right parenthesis of the FORMAT has been sensed. Attempting to read or write more characters on a record than are or can be physically present does not cause a new record to be begun; during output operations the extra characters are lost and during input operations they are treated as blanks.

3) During an input operation, processing of an input record is terminated whenever a slash specification or the final right parenthesis of the FORMAT is sensed, or when the FORMAT processor requests an item from the list and no list items remain to be processed. Construction of an output record terminates, and the record is written on the same conditions.

4) Every time a conversion specification (i.e., D, F, E, I, Z or A specification) is to be processed, the FORMAT processor requests a list item. If one or more items remain in the list, the processor performs the appropriate conversion and proceeds with the next field specification. If the next specification is one that does not require a list item (i.e., H, X or /), it is processed whether or not another list item exists. Thus, for example, the statement

```
      WRITE (6,12)
   12 FORMAT (///4HABCD)
```

would produce three blank records and one record containing ABCD before reaching the final right parenthesis. When there are no more items remaining in the list and the final right parenthesis has been reached or a conversion specification has been found, the current record is terminated, and control is passed to the statement following the READ or WRITE statement that initiated the input/output operation.

5) When the final right parenthesis of a FORMAT statement is encountered by the FORMAT processor, a test is made to determine if all list items have been processed. If the list has been exhausted, the current record is terminated and control is passed to the statement following the READ or WRITE statement that initiated the input/output operation. However, if another list item is present, an additional record is begun, and the FORMAT statement is rescanned. The rescan takes place as follows:

   a. If there are no parenthesized groups of specifications within the FORMAT statement, the entire FORMAT is rescanned.

   b. However, if one or more parenthesized groups do appear, the rescan is started with the group whose right parenthesis was the last one encountered prior to the final right parenthesis of the FORMAT statement. In the following example, the rescan begins at the point indicated.

```
FORMAT(3X,(F7.2,A5),(X,3HABC(3I4,(E15.7//),A3)),E20.8,3HXYZ)
                    ↑                         ↑              ↑
                 rescan                  closing        final right
                 begins                  parenthesis    parenthesis
                 here                    of internal    of FORMAT
                                         group
```

   c. If the group at which the rescan begins has a repeat count (r) in front of it, the previous value of the repeat count is used again for each rescan.

6) Each list item to be converted is processed by one specification or one iteration of a repeated specification.

8.2.6 Field Termination by Comma

An additional feature has been introduced for input of numeric input strings by E, F or I format specification. The numeric input string can be terminated by a comma (,) relieving the user of the concern of editing his data in proper columns.

Example:

```
      READ (3,10) K,X,Y
10 FORMAT (I10,E16.8,F14.2)
```

The input data string can be typed as

```
135, 1.23E+6, 235.,
```

where the comma will terminate the field of the input string being processed.

Warning:

A trap is best illustrated by the following example:

```
      READ (3,10) I1,I2,I3
10 FORMAT (3I4)
```

If the input string is typed as follows

```
23,6420,16,
```

the internal values of the variables will become

```
I1 =    23
I2 =  6420
I3 =     0    NB!
```

The explanation is that as the first comma terminates the first field, the second comma will terminate the third field because the second number of four digits will terminate the second field (I4).

Example:

If ,,, is typed in the above example, the result will become:

I1 = I2 = I3 = 0

The presence of a carriage return "CR" in the input string will have the same effect as a comma, as it will terminate the field.

8.3 Binary Input/Output

The binary transmission mode merely transports bit-pattern from one place to another, e.g., from magnetic tape into computer memory or the reverse.

One half-word (8 bits) is moved at a time. Two neighbour half-words are placed side by side in one memory location. The order of the half-words is preserved during the transfer.

8.4 Standard Format Input/Output

8.4.1 "Free" Format Input

The FIO-system includes an input option that relieves the programmer from the difficulty of input format description. The statement that causes this option is

INPUT (m) list

where m is a logical unit number. The elements of the list determine which type of format specification the conversion will follow. An integer in the list will cause the input conversion to follow an I16 specification. A real in the list will cause the input conversion to follow E16.0.

Each field in the input data string is terminated either by a comma (,) or a carriage return (CR) (see Section 8.2.6). Note that a CR should not be preceded by a comma if the list is not exhausted, as the CR then will terminate the next field and have the same effect as a comma followed by a comma. A maximum of twenty data fields can be input in one record (line).

Example:

```
INPUT (3) I1,I2,X

The data string can be input as
12, 526, 1.25E-6 CR
or
12 CR LF
526, 1.25E-6 CR
```

Note:
The conversion will always follow the rules for I16 or E16.0 specification.

Example:

```
INPUT (3) X
```

Typing: 3269, without a decimal point will cause the internal value to become: X = 3269.

8.4.2 Standard Format Output

Standard output formats are I16 for integers, E16.8 for reals.

The output appears with four numbers on each line, if there is output sufficient to fill a line.

This type of output is effected by the statement

OUTPUT (m) list

where m is device number.

8.5 Format Control

The first character in a formatted output record is always used for format control to direct the line printer. The table below shows the reactions of the printer on different characters in the first position.

Character	Reaction
Blank	Simple record shift
0	Double record shift
1	New page
+	Same record as before
$	Append actual record to last one with no CR/LF

All other characters in the first position act as blanks and are skipped.

9 DIRECTIONS FOR USE

9.1 NORD STANDARD FORTRAN System for TSS/SINTRAN III

9.1.1 The Compiler

The FORTRAN compiler may be recovered from the TSS Utility Command Processor (@) by typing

FTN↵

Initially the compiler will run into its command processor (outputs $ on the Teletype). In this mode it accepts the following commands terminated by carriage return:

REFMAP — A reference map containing all identifiers used along with their (relative) addresses will be printed on the list file/device after each compiled program unit.

DEBUG — In this mode the compiler will generate the additional code necessary to run the program supervised by the FORTRAN Debugging System.

N-TEN — Special NORD-10 code will be generated (default mode on NORD-10 TSS).

N-ONE — Ordinary NORD-1 code will be generated.

CLC <octal address>

In front of each listed statement the corresponding core address will be printed out with the specified number regarded as base address.

COM <source file>, <list file>, <object file>

This command will start the compilation with the specified file/device combination. The files may be specified by:

1) Octal file numbers (cfr. Appendix G). Except from file 1 (Teletype) and 100 (scratch) these files/devices must be opened/reserved from TSS.

2) Symbolic file/device names. (Cfr. ND Time-sharing System.)

Example:

```
@ FTN↵
$ CLC␣65000↵
$ REFMAP↵
$ COM C-R, L-P, "OBFILE"↵
```

The compilation terminates when an EOF statement (necessary!) is encountered and the compiler returns to its command processor ($).

D

Deposit new value (octal) into the specified address. Type the address terminated by /, then the contents of the location will be printed, and then the user may type the new value he wants to deposit, or just give carriage return if no change is wanted.

Example:

Deposit the instruction JMP*-1 (124377) into location 302

```
L*D 302/125000 124377
L*
```

E <file>

Examine contents of locations.

This command will print the contents of the specified locations in octal format on the specified logical file.

Example:

```
L*E 1 10 15↵
000010/001234
001235
000000
000000
000014
000015
L*
```

In this example the contents of locations 10 through 15 are printed on the device with the logical device number 1.

This command is equal to the MAC command)PRINT.

F

Fix loader symbol table and set lower bound equal to current location.

I

Define new size of loader symbol table.

Example:

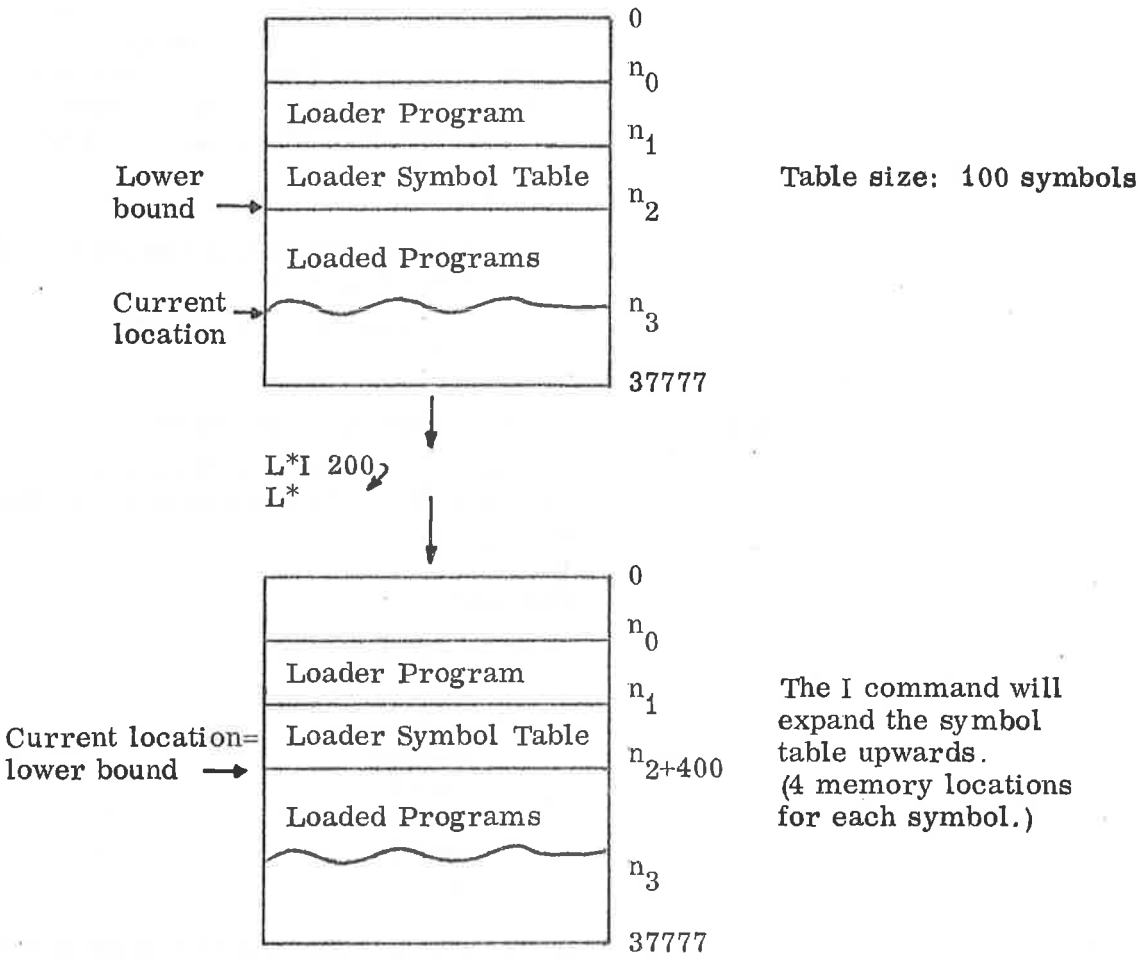
```
L*I 100↵
L*
```

The loader symbol table will hold 100 (octal) symbols in this example.

The loader symbol table is placed immediately after the loader program, and will be expanded upwards. This command will also set current locations equal to the new lower bound.

We will advise the users to use the I command before any program units are loaded.

CORE LAY OUT



Figure

Note: Take care not to expand the loader symbol table into loaded programs.

L — Set start load address.

Example:

```
L*L 5000
```

In this example the next program to be loaded will be loaded from location 5000 (octal) and upwards.

M <file> — Manual mode.

Load one program unit (until END) from the file specified (symbolic or logical), and return control to the command processor.

N <file>

List undefined symbols.

This command will list all undefined symbols in the loader symbol table on the specified logical file.

Example:

```
L*N 5↵
   PER U006000
PRINT U006010
 OLER U006070
L*
```

These three lines will be printed by the line printer

R

Reset loader.

S

Start execution of the loaded program.

U

Define upper address for loader area (upper bound).

Example:

```
L*U 70000↵
L*
```

In this example upper bound will be set to address 70000.

W <file>

Write defined symbols.

This command will list all defined symbols in the loader symbol table on the specified logical file.

Example:

```
L*W 3↵
  TOR = 005000
 NILS = 005010
NILS2 = 005011
  ALF = 005400
@
*  : 007600   <-
C  : 070000   <-
L*:
```

These lines will be punched on paper tape

Value of current location

Lower address of common area

This symbol list, until the character @, may be read into the MAC assembler's symbol table, and used for linking of binary programs and BRF programs, or for debugging purposes.

X <file>

Define symbols.

This command reads symbols and values from the specified logical file into the loader symbol table. The symbol list must be terminated by the character * or @.

Example:

```
L*X 1↵
SYMBL = 001000
   PER = 050000
  SINU = 050100
*
L*
```

} Read from the Teletype

This command may also read a symbol list produced by the MAC assembler's)LIST command.

Y — Define only undefined symbols.

If the command X is used after the command Y, only undefined symbols will be defined by the command X and the other symbols will be skipped.

Example:

```
L*N 1↵
SYMBL U004000
   PER U005000
  NILS U005500
L*W 1↵
* : 006000
C : 077777
L*Y
L*X 1↵
SYMBL = 010000
   OLE = 123456
*
L*W 1↵
SYMBL = 010000
* : 006000
C : 077777
L*N 1↵
   PER U005000
  NILS U005500
L*
```

In this example the symbol OLE will not be defined by the command X, because it was not undefined in loader symbol table.

Z<file> Undefined symbol.

This command will read symbols from the specified logical file, and makes them undefined in the loader symbol table. The symbol list must be terminated by the character @ or *. In the location where the symbol will be undefined, the loader will deposit the value -1.

Example:

```
L*Z 1,
SYMBL = 10
   OLE = 20
*
L*
```

In this example the symbol SYMBL will be undefined in address 10 and the symbol OLE in address 20, and the contents of address 10 and 20 will be -1.

If we now use the N command, the result will be:

```
L*N 1,
SYMBL U000010
   OLE U000020
L*
```

The commands X, Y and Z are not standard, but they are available as an option.

H Exit from the loader.

O <file name> , <decimal file number>, <mode>

The O command is used to open a file referenced by the user program and to assign specified device number to that file.

<file name> Name of file.

<decimal file number>

File number used by the user program to reference the file.

<mode> R, W, RX or WX with the following meaning:

R - open for read sequential
W - open for write sequential
RX - open for read random
WX - open for read/write random.

The O command may also be used to change the device number of a unit record device.

Example:

L*O LINE PRINTER, 123, W

The FORTRAN statement

OUTPUT (123)

will now give output to the line printer.

9.1.3 Map of Memory after Loading

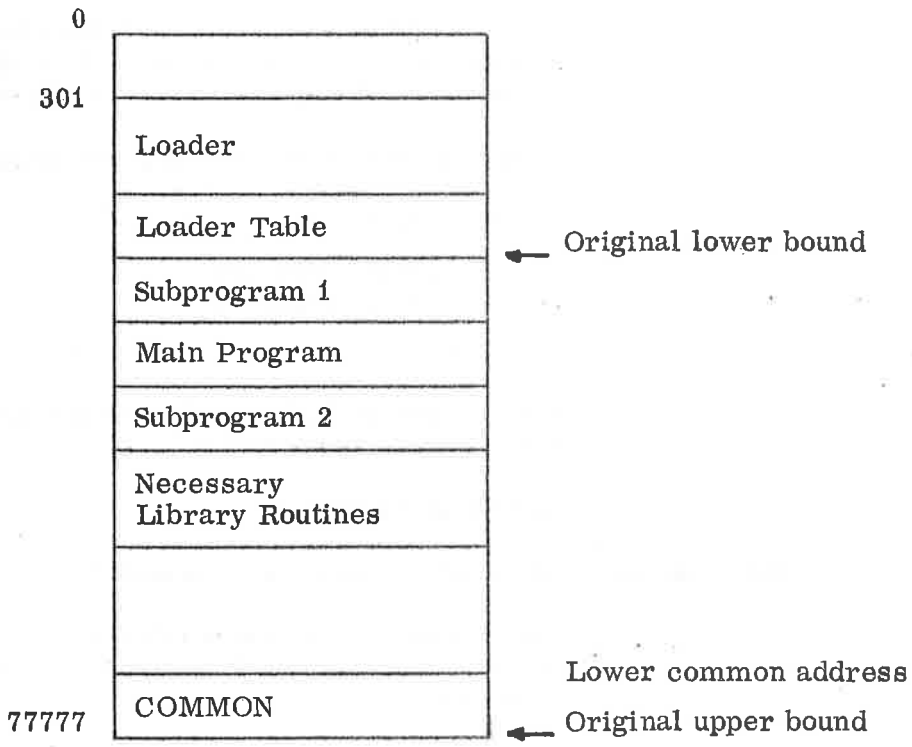
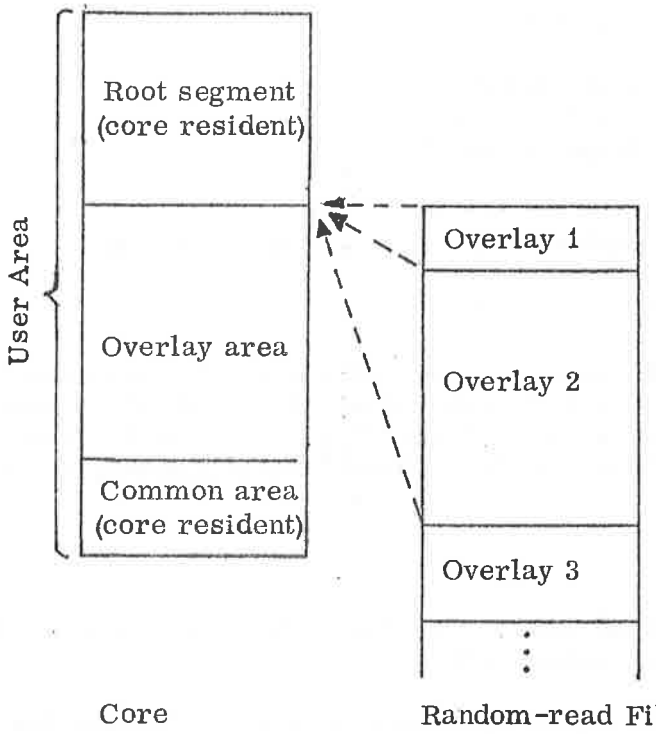


Figure 9.2

9.2 Overlay Segmentation of FORTRAN Programs

The overlay structure consists of a main program, referred as the root segment, and one level of associated overlay segments.



The root segment and the common area reside in memory throughout the entire execution, while the overlays reside on a random-read file. When any of the overlay subprograms are called from the root segment, the run-time system will load the appropriate overlay (if not already present) into the core overlay area.

Thus the root segment may reference any other root segment or overlay subprogram, while an overlay subprogram may only reference subprograms in its associated overlay or in the root segment.

The root segment is loaded into core in the usual way and ahead of any overlay. An overlay is specified from the loader by

L* #<name1>,<name2>, ...,<nameN>

where the names refer to subprograms called from the root segment.

When this command is given, the specified subprograms can be loaded from a BRF file. It is recommended to keep the overlay subprograms on a separate BRF file compiled in library mode (ref. Section 9.1.1). In this way the specified set of subprograms may be selected and loaded into the overlay independent on the compilation sequence.

In the following example the root segment is compiled into the file ROOT:BRF, and the subprograms into LIBSUB:BRF (in library mode) in the sequence SUBR1, SUBR2, SUBR3, SUBR4. To generate a program system with SUBR1, SUBR4 on overlay 1 and SUBR2, SUBR3 on overlay 2, the following command sequence will apply:

```
L*A ROOT:BRF
L*# SUBR1, SUBR4
L*A LIBSUB:BRF
L*# SUBR2, SUBR3
L*A LIBSUB:BRF
L*
```

The overlay read-only file may be specified by the

G <file name>

When used before the previous mentioned overlay command (#), the loader and runtime system will use the specified file instead of the default scratchfile 100. Thus, it is possible to dump and recover a generated program system without bothering saving the scratchfile contents.

Note:

1) A special loader, OVERLAY-BRL, is required to load and execute with overlays.

2) The debugging option cannot be used in connection with overlays.

10 NORD STANDARD FORTRAN DEBUGGING OPTION

By the debugging facility the user is able to execute his program while tracing, stepping or breaking through it. Variables may be examined and modified whenever wanted, like an interactive execution on assembly level.

10.1 The Compilation and Load Procedures

10.1.1 Compilation

If the program should be executed in debugging mode, the DEBUG command must be given before the compilation ($DEBUG).

10.1.2 Loading

In addition to the ordinary run-time system the debugging supervisor must be present prior to the execution. This supervisor is called 8DBUG and occupies some 1.5K of storage. By typing S from the loader, the control is transferred to 8DBUG.

10.2 Syntax of the Command

When the debugging supervisor prints an & on the Teletype, it is ready to accept a command. The available commands (along with possible arguments) must be typed on the same line as the & and terminated by a carriage return.

Space has delimiting effect, but more than one in a sequence are ignored.

10.2.1 Syntax of the Arguments

An argument may be

1) A decimal number.
2) One or two statement specifications.
3) One or more symbolic FORTRAN variable names.

10.2.1.1 Statement Specifications

The general syntax is

<program unit name>,<statement number>+<displacement>

However, if the referenced statement belongs to the same unit as the next statement of execution, the unit name may be omitted:

< statement number > + < displacement >

Furthermore, if no numbered statement precedes the referenced one, the statement number is dropped.

<program unit name> + <displacement>

A zero displacement may be omitted in the specification. All displacements must be positive.

Examples:

Specification:	Comment:
SUBR, 100+2	Two statements beyond that of label 100 in SUBR.
10+5	Five statements beyond that of label 10 in actual unit.
PROG+2	Third statement of PROG.
PROG2,4	Statement with label 4 in PROG2.

10.2.1.2 Specifying FORTRAN Variable Names

The general syntax is

<program unit name> , <name>

If the variable belongs to the same unit as the next statement of execution, the program unit name may be omitted. Arrays may be indexed with constants as subscripts (array elements).

Examples:

```
OLE, A
  B
SUBR, ARRAY (26)
ARR (1,1,1)
```

10.3 The available Commands

10.3.1 TRACE <statement specification><statement specification>

The flow of control of the FORTRAN program may be examined through all statements executed (TRACE <carriage return>) or through one or more trace areas, each specified by a lower and an upper bound.

During execution a reference to each passed statement will be printed out. These references are preceded by the word TRACE enclosed in brackets.

Example:

```
&TRACE OLE,10+1 OLE,100
```

10.3.2 BREAK <statement specification>

When the specified statement is reached, the execution will halt and the control will be transferred to the debugging supervisor. The break is performed before the specified statement is executed.

Example:

```
&BREAK OLE,10+2
```

10.3.3 COND <variable name><relational operator><constant>

When/if specified condition is true, the control will be transferred to the debugging supervisor.

The specified variable must be of type integer or real only.

All the FORTRAN standard relational operators, i.e. .LT.,.LE.,.EQ., .NE.,.GE. and .GT. are permitted. If the specified condition causes a break, it will be reset automatically (contrary to the BREAK command).

Examples:

```
&COND SUBR,A(2) .EQ. 4.5↵
&COND I .GT. 6↵
```

10.3.4 DISPLAY <variable name><variable name>... etc.

In trace mode the specified variable names will be printed out followed by a colon and their current values.

10.3.5 BOUND <array name> (<index1>, ... <indexn>)

The array should be specified with the greatest indices permitted. If the array is accessed beyond this range, a message will be given and the control will be transferred to the debugging supervisor.

Example:

```
&BOUND SUBR,ARR (4,4)↵
```

10.3.6 RESET

RESET may be used in front of the TRACE, BREAK, DISPLAY and BOUND commands with or without arguments (no arguments of RESET BREAK and BOUND). Its effect is to delete an earlier given argument of the four commands listed.

10.3.7 WHERE (or *)

WHERE prints a reference on the Teletype to the next statement of execution.

10.3.8 DEVICE <logical device number>

By this command the user may specify the output device of trace information and display parameters.

10.3.9 > (Step Command)

The next statement will be executed according to the dynamic flow of control of the program. Thus this command decreases the speed of execution only and the track is never lost.

10.3.10 CONTINUE (or C)

The execution will continue from the next statement.

10.3.11 NEST

This command displays the routine nesting in the format:

```
&NEST,
<name of present unit>
<name of caller>
      :
      :
<name of main program>
```

10.3.12 LDR

The control is transferred to loader.

10.3.13 EXIT

Exits to TSS/SINTRAN III.

10.4 Examination of variable Values

When the supervisor prints the character &, the values of single or subscripted variables may be examined and possibly modified. This may be obtained by typing the name (cfr. Section 10.2.1.2) of it followed by a slash:

<variable name>/

The value, which will appear on the right side of the slash, may be changed by typing a left arrow (←) along with the new value and terminating with carriage return.

Examples:

& A / 2.300000E+100← 4.5↵
& IARR(10) / -4← -3↵
& FUNC, B / 10↵

APPENDIX A

CODING PROCEDURES

Statements

FORTRAN coding forms contain 80 columns; the characters of the language are written, one per column, in columns 7 through 72. Statements longer than 66 columns may be carried to the next line by using a continuation designator. No more than one statement may be written on a line. Blanks may be used freely in FORTRAN statements to improve readability. Blanks are significant only in Hollerith fields of format specification nH or '.....'.

Statement Identifiers

Any statement may have an identifier but only statements referred to elsewhere in the program require identifiers. A statement identifier (also called a statement label or statement number) is a string of from one to five digits, 1 to 32767, in columns 1 through 5. The value of the identifier is not significant, but it must be positive. Leading zeroes are ignored; 1, 01, 001, 0001 are equivalent forms. Zero is not a statement identifier. In any given program unit each statement identifier must be unique.

Lines

A line is a string of maximum 72 characters from the FORTRAN character set. Lines may be initial, continuation, comment or end. In an initial line, the first line of any statement, column 6 must be zero or blank. Only an initial line may have a statement identifier in columns 1 through 5. If there is no statement identifier, columns 1 through 5 are blank. A statement with statement number must be blank in column 6.

If a statement occupies more than one line, all subsequent lines must have a FORTRAN character other than zero, or blank in column 6. Every program and subprogram must be terminated by an end line indicating that the written description of the program unit is complete.

Comments

A comment line is designated by the letter C in column 1, and contains comment information in columns 2 through 72. Comment information is a convenience to the programmer; it appears in the source program but is not translated into object code. Continuation is not permitted; each line of comments must be preceded by the C designator.

Carriage Return (CR)

Carriage return is used for termination of a line. It may occur anywhere on the line from column 1 to column 80. If the source program is punched on cards, column 81 will be CR and column 0 will be LF (line feed). Source programs typed on paper tape must start each line with line feed and terminate it with carriage return. Dummy lines and blank cards are ignored. Any occurrence of characters not included in the FORTRAN set will result in an error message and the rest of the statement will be skipped.

Columns 73 to 80 may be used for identification. It is illegal to use carriage return within the identification. If attempted, the line will terminate and a new line will be started containing the rest of the identification.

APPENDIX B

STATEMENTS OF NORD STANDARD FORTRAN

Statement Form	N/E	Page
ASSIGN	E	5-1
BACKSPACE u	E	7-6
BLOCK DATA	N	4-10
CALL s	E	6-8
CALL s($a_1, \ldots, a_n$)	E	6-8
COMMON/x_1/a_1/.../x_n/a_n	N	4-3
CONTINUE	E	5-8
DATA $k_1/d_1/, \ldots, k_n/d_n/$	N	4-9
DIMENSION $v_1(i_1), \ldots, v_n(i_n)$	N	4-2
DO n i = m_1, m_2	E	5-4
DO n i = m_1, m_2, m_3	E	5-4
END	E	5-9
ENDFILE u	E	7-6
EQUIVALENCE $(k_1), \ldots, (k_n)$	N	4-6
EXTERNAL $v_1, \ldots, v_n$	N	6-7
FORMAT $(q_1 t_1 z_1 \ldots t_n z_n q_n)$	N	8-2
t FUNCTION f ($a_1, \ldots, a_n$)	N	6-3
t may be any of the following: INTEGER REAL DOUBLE PRECISION COMPLEX LOGICAL		
GO TO k (unconditional GO TO)	E	5-1

N = non-executable, E = executable

Cont.

Statement Form		N/E	Page
GO TO i, (k_1, k_2, . . . , k_n)	(assigned GO TO)	E	5-2
GO TO (k_1, . . . , k_n), i	(computed GO TO)	E	5-2
IF (e) k_1, k_2, k_3	(arithmetic IF)	E	5-3
IF (L) s	(logical IF)	E	5-4
INPUT (i) L	(standard format)	E	7-3
OUTPUT (i) L	(standard format)	E	7-2
OUTPUT (i, a) L	(RT-FORTRAN)	E	9-14
PAUSE		E	5-9
PAUSE n		E	5-9
PROGRAM		N	6-1
READ (i, n) L	(formatted)	E	7-2
READ (i, n)	(formatted)	E	7-2
READ (i) L	(binary)	E	7-4
READ (i)	(binary)	E	7-4
RETURN		E	6-11
REWIND i		E	7-6
STOP		E	5-9
STOP n		E	5-9
SUBROUTINE s		N	6-7
SUBROUTINE s (a_1, . . . , a_n)		N	6-7
t v_1, . . . , v_n	(type statement)	N	4-1
t may be any of the following: INTEGER DOUBLE INTEGER REAL DOUBLE PRECISION COMPLEX LOGICAL			

Cont.

N = non-executable, E = executable

Cont.

Statement Form		N/E	Page
v = e	(arithmetical replacement)	E	3-5
WRITE (i,n) L	(formatted)	E	7-1
WRITE (i,n)	(formatted)	E	7-1
WRITE (i) L	(binary)	E	7-3

N = non-executable, E = executable

APPENDIX C

LIBRARY FUNCTIONS OF NORD STANDARD FORTRAN

External Functions

External Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Exponential	e^a	1	EXP	Real	Real
		1	DEXP	Double	Double
		1	CEXP	Complex	Complex
Natural Logarithm	$\log_e(a)$	1	ALOG	Real	Real
		1	DLOG	Double	Double
		1	CLOG	Complex	Complex
Common Logarithm	$\log_{10}(a)$	1	ALOG1	Real	Real
			DLOG10	Double	Double
Trigonometric Sine	sin(a)	1	SIN	Real	Real
		1	DSIN	Double	Double
		1	CSIN	Complex	Complex
Trigonometric Cosine	cos(a)	1	COS	Real	Real
		1	DCOS	Double	Double
		1	CCOS	Complex	Complex
Hyperbolic Tangent	tanh(a)	1	TANH	Real	Real
Square Root	$(a)^{1/2}$	1	SQRT	Real	Real
		1	DSQRT	Double	Double
		1	CSQRT	Complex	Complex
Arctangent	arctan(a)	1	ATAN	Real	Real
		1	DATAN	Double	Double
	$\arctan(a_1/a_2)$	2	ATAN2	Real	Real
		2	DTAN2	Double	Double
Remaindering*	$a_1 \pmod{a_2}$	2	DMOD	Double	Double
Modulus		1	CABS	Complex	Real

* The function DMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2]\, a_2$, where $|\alpha|$ is the integer whose magnitude does not exceed the magnitude of α and whose sign is the same as the sign of α.

Intrinsic Functions*

Intrinsic Function	Definition	Number of Arguments	Symbolic Name	Type of Argument	Type of Function
Absolute Value	$\|a\|$	1	ABS IABS DABS	Real Integer Double	Real Integer Double
Truncation	Sign of a times largest integer $\leq \|a\|$	1	AINT INT IDINT	Real Real Double	Real Integer Integer
Remaindering*	$a_1 \pmod{a_2}$	2	AMOD MOD	Real Integer	Real Integer
Choosing Largest Value	Max $(a_1, a_2, \ldots)$	≥ 2	AMAX0 AMAX1 MAX0 MAX1 DMAX1	Integer Real Integer Real Double	Real Real Integer Integer Double
Choosing Smallest Value	Min $(a_1, a_2, \ldots)$	≥ 2	AMIN0 AMIN1 MIN0 MIN1 DMIN1	Integer Real Integer Real Double	Real Real Integer Integer Double
Float	Conversion from integer to real	1	FLOAT	Integer	Real
Fix	Conversion from real to integer	1	IFIX	Real	Integer
Transfer of Sign	Sign of a_2 times $\|a_1\|$	2	SIGN ISIGN DSIGN	Real Integer Double	Real Integer Double
Positive Difference	a_1-Min (a_1, a_2)	2	DIM IDIM	Real Integer	Real Integer
Obtain Most Significant Part of DP Argument		1	SNGL	Double	Real
Obtain Real Part of Complex Argument		1	REAL	Complex	Real
Obtain Imaginary Part of Complex Argument		1	AIMAG	Complex	Real
Express Single Prec. Argument in DP Form		1	DBLE	Real	Double
Express Two Real Arguments in Comp. Form	$a_1 + a_2\sqrt{-1}$	2	CMPLX	Real	Complex
Obtain Conjugate of a Complex Argument		1	CONJG	Complex	Complex

* The function MOD or AMOD (a_1, a_2) is defined as $a_1 - |a_1/a_2|\, a_2$, where $|x|$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x.

Simulating Routines*

Simulating Routine	Definition	Number of Arguments	Type of	
			Argument	Function
8AXA	R**R	2	Real	Real
8AXI	R**I	2	Real, Integer	Real
8DXI	DP**I	2	Double, Integer	Double
8IXI	I**J	2	Integer	Integer
8DIV	I/J	2	Integer	Integer
8FIX		1	Real	Integer
8IAD	DI1+DI2	2	Double integer	Double integer
8ISB	DI1-DI2	2	Double integer	Double integer
8IMU	DI1*DI2	2	Double integer	Double integer
8IDV	DI1/DI2	2	Double integer	Double integer
8IDI		1	Double integer	Integer
8IDR		1	Double integer	Real
8RID		1	Real	Double integer
8CAD	C1+C2	2	Complex	Complex
8CSB	C1-C2	2	Complex	Complex
8CMU	C1*C2	2	Complex	Complex
8CDV	C1/C2	2	Complex	Complex
8CXI**	C**I	2	Complex, Integer	Complex

* Simulating routines are automatically activated by expressions shown under the definition.

** 8CXI makes use of SQRT, 8AXI, ATAN, SIN and COS.

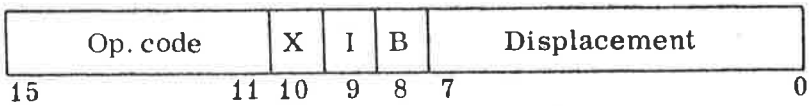
Cont.

Simulating Routine	Definition	Number of Arguments	Type of	
			Argument	Function
8DAD	DP1+DP2	2	Double	Double
8DSB	DP1-PD2	2	Double	Double
8DMU	DP1*PD2	2	Double	Double
8DDV	DP1/DP2	2	Double	Double

APPENDIX D

NORD WORD STRUCTURE

Instruction word



One instruction word always occupies one location, 16 bits, of core memory. The operation code occupies the five most significant bits (11 - 15), and specifies one of 32 instructions.

For memory reference instructions bits 0 - 10 are used to specify the address of the instruction. The instructions which do not have an address, use these bits to further specifications. Bits 8, 9, and 10, called ,B I and ,X are used to control the address computation.

The displacement is an 8 bit signed number ranging from -128 to +127, using two's complement for negative numbers and sign extension to produce a 16 bit number.

Data word

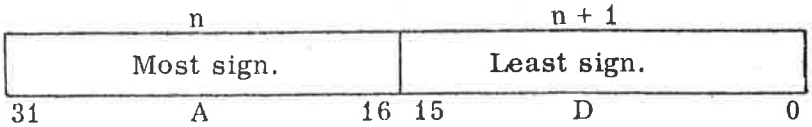
Three basic types of data words exists:

a) Single length numbers:

A 16 bit number which occupies one memory location. Representation of negative numbers are in two's complement. Range as integers: $-32768 \leq x \leq 32767$

b) Double length numbers:

A 32 bit number which occupies two consecutive locations in memory, and where negative numbers also are in two's complement.



A double word is always referred to by the address of its most significant part. Normally a double word is transferred to the registers so that the most significant part is contained in the A-register and the least significant in the D-register. Range as integers: $-2\ 147\ 483\ 648 \leq x \leq 2\ 147\ 483\ 647$

c) Floating point numbers:

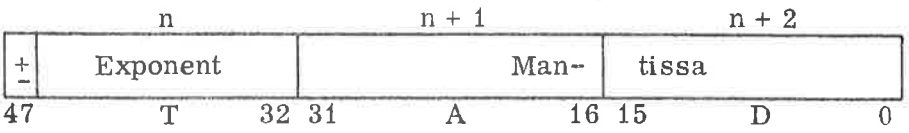
The data format of floating point words is 32 bits mantissa magnitude, one bit for the sign of the number and 15 bits for a signed exponent.

The mantissa is always normalized, $0.5 \leqslant \text{mantissa} < 1$; for all non-zero numbers bit 31 equals one. The exponent base is 2. The exponent is biased with 2^{14}, i.e. 40000_8 is added to the actual exponent, so that a standardized floating zero contains zero in all 48 bits.

In core store one floating point data word occupies three 16 bit core locations, which are addressed by the address of the exponent part.

n	exponent and sign
n+1	most significant part of mantissa
n+2	least significant part of mantissa

In CPU registers bits 0 - 15 of the mantissa are in the D register, bits 16 - 31 in the A register, and bits 32 - 47, exponent and sign, in the T register. These three registers together are defined as the floating accumulator.



The accuracy is 32 bits or approximately 10 decimal digits, any integer up to 2^{32} has an exact floating point representation. The range is:

$$2^{-16384} \cdot 0.5 \leqslant |x| < 2^{16383} \cdot 1 \text{ or } x = 0$$

or

$$10^{-4931} \leqslant |x| < 10^{4931}$$

Examples (octal format):

	T	A	D
0:	0	0	0
+1:	040001	100000	0
-1:	140001	100000	0

Any other data word format than those three described here may be programmed. These three data word formats have corresponding instructions which make these formats easy and natural to use. It is also rather easy to program data word formats using one bit data word (logical variables) and 8 bit data word (character byte).

In FORTRAN, two additional data words are used:

d) Double precision numbers:

The data format of double precision words is 80 bits mantissa magnitude, one bit for the sign of the number and 15 bits for the signed exponent. The mantissa is always normalized, $0.5 \leq \text{mantissa} < 1$, and for all non-zero numbers bit 79 equals one. The exponent base is 2, the exponent is biased with 2^{14}, so that a standardized double precision zero contains zero in all 96 bits.

In core store one double precision data word occupies six 16 bit core locations, which are addressed by the address of the exponent part.

n	exponent and sign
n+1	most significant part of mantissa
n+2	mantissa
n+3	mantissa
n+4	mantissa
n+5	least significant part of mantissa

The accuracy is 80 bits or approximately 24 decimal digits, any integer up to 2^{80} has an exact double precision representation.

The range is the same as for floating point numbers.

e) Complex numbers:

The data format of a complex number is two subsequent floating point words.

In core store one complex number occupies six 16 bit core locations which are addressed by the address of the exponent part of the real part.

n	exponent and sign of real part
n+1	most significant part of mantissa of real part
n+2	least significant part of mantissa of real part
n+3	exponent and sign of imaginary part
n+4	most significant part of mantissa of imaginary part
n+5	least significant part of mantissa of imaginary part

APPENDIX E

SYSTEM DIAGNOSTICS

Compiler Error Messages

The error message will be written on either the line printer or the Teletype, depending on which is specified as the listing device. If the user has requested a listing of the program, error messages will be printed on the line following the erroneous line and, in certain cases, on the next line thereafter. The error messages are selfexplanatory and are printed in the following format:

***ERROR IN <subprogram unit name><label>+<displacement><error text>

where label denotes last statement number or 0 if none are encountered yet. Displacement denotes the number of statements beyond the labeled one in which the error occurred.

Loader Error Messages

The loader error messages are selfexplanatory.

FORTRAN Formatting Error Messages

Error number	Meaning	Type of error F/I
71	Illegal character in format	F
72	Parantheses nested deeper than 5	I
73	Attempt to fetch character beyond format	F
74	Attempt to store character beyond format	F
75	-	
76	Argument error unidentified type specification (system error)	F
77		
78		
79		
80	Output record exceeds 134 characters	I
81	Format requires a greater input record	I
82	Input record exceeds 134 characters	I
83	Wrong parity in input field	I
84	Bad character in input field	I
85	Integer overflow	I
86	Real overflow on input	I
87	Real underflow on input	I
88	Real overflow on output	I
89	System error	F
90	Too big input record	F

F = Fatal, I = Informative

Arithmetical Library Error Messages

The error message is written as a combination of letters, for instance:

ddddd RUN ERR CH

This means that COSH erred in the neighbourhood of core address ddddd.

In RT-FORTRAN the error looks like

RUN ERR CH rrrrr

Here, rrrrr means the name of the RT-program.

The error messages are:

AA	Error in 8AXA 8AXA was called with negative base. Result set to zero. Overflow in 8AXA. Result set to 1.0E99.
AI	Error in 8AXI Base equal to zero and exponent negative. Result set to 1.0E99.
AT	Error in ATAN2 Both arguments equal to 0.0. Result set to 0.0.
CH	Error in COSH Argument greater than 2^{14}. Result set to 1.0E99.
CO	Error in COS Argument greater than 2^{14}. Result set to 0.0
DI	Error in 8DIV Second argument equal to 0. Result set to ± 32767, depending on sign of first argument.

EX	Error in EXP Argument greater than 2^{14} ln2. Result set to 1.0E99.
GO	Argument error in a computed or assigned GO TO statement. Program returns to first label in the list.
IX	Error in 8IXI Overflow in result. Result set to 32767.
LN	Error in ALOG, ALOG1, ALOG2 Argument less or equal to 0. Result set to -1.0E99.
SH	Error in SINH Argument greater than 2^{14}. Result set to SIGN(X) · 1.0E99.
SI	Error in SIN Argument greater than 2^{14}. Result set to zero.
SQ	Error in SQRT Argument less than zero. Result set to zero.

APPENDIX F

I/O DEVICE NUMBERS

Device Number	Device
0	Dummy
1	Teletype 1
2	Paper tape reader
3	Paper tape punch
4	Card reader
5	Line printer
6	Not used

APPENDIX G

MIXED NORD STANDARD FORTRAN AND ASSEMBLY ROUTINES

The NORD STANDARD FORTRAN Run-time System has been designed to allow an extensive use of mixed FORTRAN assembly systems. No special heading format of the assembly routines is necessary, but there exist some restrictions upon the use of the B-register.

Main Program in Assembly

No restrictions.

Subprograms in Assembly

Calling assembly subroutines/functions from FORTRAN, the value of the B register by leaving the subprogram must not differ from the entering value. (System value.) Moreover, no locations in the B field ($B - 200_8$ through $B + 177_8$) must be changed by the subprogram.

Parameter Access in Subprograms

When entering any assembly subprogram, the A register points to a string of the actual parameter addresses (if any).

Access of Common Variables

)9ADS

This MAC command is used to generate addresses of LABELED COMMON variables of a FORTRAN program. Two symbols separated by blank or plus sign have to follow the command. For example,

```
PER, )9ADS ES FACIL      % PER WILL CONTAIN THE ADDRESS
                         % OF ES + THE VALUE OF FACIL
```

The first symbol must correspond to a COMMON label declared in the FORTRAN program. A blank COMMON is accessed by using the symbol BLANK. The second symbol is a displacement to the COMMON label, and must have been previously declared as fixed absolute. At load time the address of the COMMON label is added to the displacement.

Functions in Assembly

A function must always return with a value, and this must be contained in the central registers.

Logical functions	:	Logical value (0 or 1) in the A register.
Integer functions	:	Value in the A register.
Real functions	:	Value in the T-A-D registers.
Double precision and complex functions	:	These are special cases where the least significant mantissa words or the imaginary part of the function value must be placed in locations B register - 172, - 171 and - 170 (extended accumulator of the calling program). As usual the most significant or real part must be contained in the T-A-D registers.

The final instruction sequence of a complex function should therefore be:

```
        LDF   IMAGPART
        STF   -172,B          %B MUST CONTAIN THE SYSTEM VALUE
        LDF   REALPART
        EXIT
```

Example of a Subprogram Structure

```
)9BEG
)9ENT SUBR

SUBR, SWAP SA DB
      STA SAVB          %SAVES B REGISTER
      -
      -
      LDF I ,B          %ACCESS OF 1. PARAMETER
      -
      LDF I N-1,B       %ACCESS OF N'TH PARAMETER
      -
      LDA SAVB
      COPY SA DB
      EXIT              %RETURNS TO FORTRAN

SAVB,0
)9END
```

Calling Sequence of Single Argument FORTRAN Library Routines

When the jump to any of these routines is performed, the user should be be aware that the locations B - 220_8 through B - 201_8 are affected to changes from the library (scratch area).

Example:

```
        SAX 1
        LDF ARG          % PIGK UP ARGUMENT
        JPL I*1 ,X
        NN1              % ANY SINGLE ARGUMENT LIBRARY
                         % FUNCTION
        -                % RETURN WITH RESULT IN ACCUMULATOR
```

Calling a FORTRAN Subprogram from Assembly

The calling sequence is explained through the following example:

```
)9BEG
)9EXT 8ENTR SUBR
-
-
JPL I  (8ENTR            %8ENTR IS A RUN TIME TRANSITION
                         %ROUTINE
       SUBR              %FORTRAN SUBROUTINE NAME
          N              %NUMBER OF PARAMETERS
       PARAM1            %ADDRESS OF 1. PARAMETER
       -
       -
       PARAMN            %ADDRESS OF N'TH PARAMETER
       -                 %RETURN, FUNCTION VALUE IF ANY
       -                 %IN ACCUMULATOR
)FILL
)9END
```

Routines Headings and Call Sequences for Assembly Communication with NORD STANDARD RT-FORTRAN

Main Program in Assembly

```
)9BEG START
)9EXT 8RTEN RTEXT

START, JPL I*R
       STACKDEMAND    % NO. OF STACK LOCATIONS
       8RTEN
       =
       JMP I*1
       RTEXT
```

Subprogram in Assembly

```
START, =
       =
       EXIT
```

For the sake of completeness, the corresponding compiled code is referred:

Main Program in RT-FORTRAN

```
START, JPL I*2
       STACKDEMAND
       8RTEN
       =
       =
       JMP I*1
       8LEAV
```

Subprogram in RT-FORTRAN

```
START, RADD DP AD1
       STACKDEMAND
       =
       =
       JMP I*1
       8LEAV
```

Subroutine call from assembly:

```
       JPL I (8ENTR
       SUBR              % SUBROUTINE ADDRESS
            N            % NUMBER OF PARAMETERS
       PARAM1            % ADDRESS OF 1. PARAMETER
       =
       PARAMN            % ADDRESS OF N'TH PARAMETER
                         % RETURN
```

Subroutine call from RT-FORTRAN:

```
JPL I-175,B          % JPL I (8ENTR
SUBR
N+1000               % NUMBER OF PARAMETERS+1000
DISCR1               % DESCRIPTOR OF 1. PARAMETER
=
=
DISCRN               % DESCRIPTOR OF N'TH PARAMETER
                     % RETURN
```

The Descriptor Word Format

A descriptor word is divided into three bytes:

- Displacement part 0-9 (10 bits)
- Parameter type 10-13 (4 bits)
- Address mode bits 14-15 (2 bits)

Paramter type bits:

Bit 13	Bit 12	Bit 11	Bit 10	Denotes
0	0	0	0	Hollerith constant
0	0	0	1	Logical single variable
0	0	1	0	Integer single variable
0	0	1	1	Double integer single variable
0	1	0	0	Real single variable
0	1	0	1	Double real single variable
0	1	1	0	Complex single variable
0	1	1	1	Unused
1	0	0	1	Logical array/logical function
1	0	1	0	Integer array/integer function
1	0	1	1	Double integer array/double integer function
1	1	0	0	Real array/real function
1	1	0	1	Double real array/double real function
1	1	1	0	Complex array/complex function
1	1	1	1	Unused

Address Mode Bits

Bit 15	Bit 14	Effective Address
0	0	* ± d
0	1	(* - d) (indirect)
1	0	Directly B-modified
1	1	Indirectly B-modified

* address of descriptor word
d displacement
( ) contents of

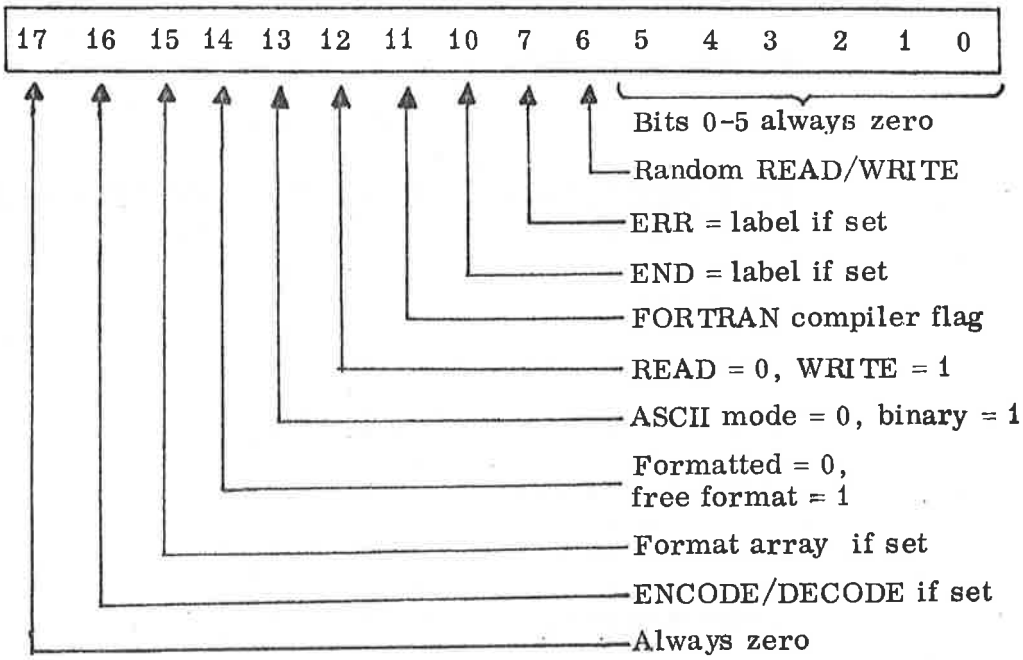
Using FORTRAN Formatted Program (FIO) from Assembly

Calling sequence:

```
JPL I (8ENTR
       8FIO          % MAIN ENTRY OF FIO
       IOINFO        % INPUT/OUTPUT INFORMATION BITS
       DEVNO         % DESCRIPTOR OF FILE/DEVICE
       FORMP         % FORMAT ADDRESS (OR 0)
       ERRL          % OPTIONAL LABEL ADDRESS OF
                     % ERR=LABEL
       ENDL          % OPTIONAL LABEL ADDRESS OF
                     % END = LABEL
JPL I -174 ,-B       % 8DATA-CALL (ADDRESS INSERTED
                     % BY 8FIO)
            N        % NUMBER OF PARAMETERS
       PARAM1        % PARAMETER DESCRIPTOR
       =
       PARAMN
JPL I -173 ,B        % 8CLSE-CALL (i/O TERMINATION)
```

IOINFO

This is a flag word with the following format:



DEVNO

This is a descriptor (no direct address) for the input/output file. The format of such descriptors are referenced above.

FORMP

This is the address of a format string, whereby a FORTRAN format description is packed as ASCII characters, two per word. The string must be enclosed by a pair of parenthesis:

```
FORMP = *-2
'(F10.2,I4)
'
```

ERRL

If bit 7 in IOINFO is set, the formatting program will exit to this address if an I/O error occurs at run time.

ENDL

If bit 10 in IOINFO is set, the formatting program will exit to this address if end-of-file is encountered during execution of a READ statement.

PARAM to PARAMN

The I/O-list represented as descriptors.

The Reserved B-field Locations used by the Run Time System (8ENTR-8LEAV)

Saved return	$B - 200_8$
Previous contents of B	$B - 177_8$
Run time stack pointer	$B - 176_8$
Address of 8ENTR	$B - 175_8$
Address of 8DATA	$B - 174_8$
Address of 8CLSE	$B - 173_8$
Extended accumulator	$B - 172_8$
Extended accumulator	$B - 171_8$
Extended accumulator	$B - 170_8$
B field 8DATA	$B - 167_8$
Pointer to second block	$B - 166_8$
Pointer to third block	$B - 165_8$
Debugging system cell	$B - 164_8$

Register Use by 8ENTR - 8LEAV

By jumping to a subroutine, the A register will point to a string of parameter addresses.

Re 8ENTR

Entry : B has its old value.

Return : B has a new value such that the 1. parameter may be accessed by e.g. LDF I-163, B etc., i.e. the A register points to B-163.

The contents of all other registers are destroyed by 8ENTR.

Re 8LEAV

Entry : B must have the same value as it had when 8ENTR (8RTEN) was left last time.

Return : B has the same value as it had when entering 8ENTR last time. Only the contents of the X register are destroyed.

APPENDIX H

NORD STANDARD FORTRAN DEVIATIONS FROM USA STANDARD FORTRAN IV X 3.9.1966

1. The following format conversion code is missing:

   G - Generalized floating point conversion.

2. Additional Features

   a) DOUBLE INTEGER type declarations.
   b) The logical operators .AND., .OR. may also operate on integers.
   c) Decimal digits allowed in STOP and PAUSE.
   d) Apostrophes may delimit Hollerith strings.
   e) T format field descriptor.
   f) Consecutive slashes in formats cause blank lines when printed.
   g) ASSIGN statement and associated ASSIGNED GOTO statement are required to be in same program unit,
   h) Optional comma in COMPUTED and ASSIGNED GOTO.
   i) All types of arithmetic assignment statements.
   j) Integer expressions in DO statements.
   k) END= in READ statements.
   l) Real and double precision DO control variables.
   m) PROGRAM statement.
   n) Many levels of parentheses in formats.
   o) Expressions in output lists.
   p) ERR= in READ /WRITE statements.
   q) Seven dimensions in arrays.
   r) Array elements may occur in STATEMENT FUNCTION definitions.
   s) A complex number may be equivalenced to two real numbers.
   t) Array names without subscripts in EQUIVALENCE.
   u) END acts like STOP or RETURN.
   v) Prints all asterisks when number exceeds field.
   w) Array name without subscript in DATA statements.
   x) Non-FORTRAN characters on COMMENT lines.
   y) ENCODE/DECODE effect in I/O statements (Ref. VII, chapter 7.1).

A/S NORSK DATA-ELEKTRONIKK

COMMENT AND EVALUATION SHEET

Publ. No.
ND-60.011.04

NORD STANDARD FORTRAN

In order for this manual to develop to the point where it best suits your needs, we must have your comments, corrections, suggestions for additions, etc. Please write down your comments on this pre-addressed form and post it. Please be specific wherever possible.

FROM _____

_____

_____

– we want bits of the future

A/S NORSK DATA-ELEKTRONIKK ØKERNVEIEN 145 OSLO 5 NORWAY PHONE: 21 73 71 TELEX: 18284